Dancing Guide: Near Realtime Audio Classification of Live Dance Music on Smartphones

Alexander K. Seewald[1]
[1]*Seewald Solutions, Lärchenstraße 1, A-4616 Weißkirchen a.d. Traun, Austria*
alex@seewald.at

Keywords: Audio Classification, Live Music, Dance Music, Smartphones, Music Genre Classification, Near-Real-Time

Abstract: Between 2008 and 2014 we developed and deployed a live music classification system, Dancing Guide, to be run on Android and iPhones mobile phones in near realtime. Although internet access was needed to send feedback and classifications to the server for statistical purposes, the music classification system also worked offline without any loss in accuracy or speed. This is essential since in most discos and dancing schools, internet access is spotty at best. During the seven years of the project, the app was available both for iPhone and Android, initially in German and English, but later – thanks to volunteer translations – also in Czech, Spanish, French, Italian, Japanese, Korean, Dutch, Polish, Portuguese, Russian and Traditional Chinese. Measured by user feedback, we achieved an accuracy of roughly 73% at a coverage of 61%. While the accuracy is comparable to initial estimates using cross-validation, the coverage is much worse. Background noise – which we did not model – or the limited feature set may have been responsible. We retrained the system several times, however performance did not always improve, so we sometimes left the previously trained system in place. In the end, the limited feature set which was initially chosen prevented further improvement of coverage and accuracy, and we stopped further development.

1 INTRODUCTION

Between 2008 and 2014 we developed and deployed a live music classification system, *Dancing Guide*[1], to be run on then-current Android and iPhones mobile phones in near-real-time as an app.[2] Although internet access was needed to send feedback and classifications to the server for statistical purposes, the model was deployed locally and the music classification system was able to work offline without any loss in accuracy or speed. This was essential since in most discos and dancing schools internet access is spotty at best.[3]

The motivation for building this system was that it is very hard for dancing beginners to find out which dance is appropriate for given live music. So we built Dancing Guide to enable dancing beginners to find out which dance is appropriate for given live music.

During the seven years of the project, the app was available both for iPhone and Android, initially in German and English, but from December 2009 onwards – thanks to volunteer translations – also in Czech, Spanish, French, Italian, Japanese, Korean, Dutch, Polish, Portuguese, Russian and Traditional Chinese. Measured by actual user feedback – surely the most desirable feedback as it is completely independent of training, validation or testing data – we achieved an accuracy of roughly 73%. We retrained the system several times, however in some cases performance remained very similar or was even worse than an earlier system so we sometimes left the previously trained system in place. In total, the app ran on 3,533 different models of phones.

We do not have reliable user data from 2008 and 2009 since there was a bug in the unique id computation and many users obtained the same ids. The number of unique users, feedbacks and feedbacks/unique user from 2010 to 2014 is later shown in Table 1. The table also shows accuracy and coverage according to user feedback, or alternatively the proportion of outputs where no dance could be recognized (a trade-off to increase accuracy where low-confidence dance music predictions are not shown to the user, i.e. $1 - \textit{Coverage}$), and the proportion of recording errors (i.e. when no audio could be recorded due to hardware or software problems, or simply the microphone being occupied by a call).

[1]Project page: https://dg.seewald.at

[2]In the beginning, we also had a working version for Nokia and Ericsson mobiles phones using Java ME (CDLC)

[3]This may have changed in the meantime.

Contrary to music identification systems who rely on exact timing of played notes and need databases containing several billions of songs, our system was designed to analyze instrumental classical dance music played by a live band or orchestra, for example at Austrian balls or in dancing schools, where music id systems such as the then current Shazam (Wang, 2003) fail.[4] In our case music identification is not a reasonable task as every live song is to some extent unique. We therefore focussed on the more feasible task of classifying music genre relevant to dance music and eventually chose these five genres.

1-5. Cha-Cha-Cha, Jive, Samba, Tango, (Viennese/Slow) Waltz

We directly analyze audio data by beat and spectral features. By this, we can recognize completely new and unknown songs which have not been imported into the song database of music id systems yet. Many adapted and improvised dance songs are within this category.

2 RELATED RESEARCH

When we started the project in 2008, there was already some research on dance music genre classification. We can still reasonably claim to have been the first to build a near-real-time dance music genre classification system that works with short segments[5] in the presence of background noise, and works for live as well as recorded dance music.

(Wang, 2003) describe an – in 2008 – state-of-the-art music identification system. He used local peaks in audio spectrograms to compute hashes from peaks within target zones, yielding two frequency components and a time shift. A density-based sampling is used to get roughly the same number of peaks for each time span. For a large database of known music, all such hashes are computed and stored. To find a sample, all peaks from the sample are searched in the database, and are checked whether the matched peaks correspond to the known time shifts. Since this method is sensitive to the time shifts which will change for every performance of live music, their method cannot be applied to the task of live music dance classification even when the underlying music is stored in the database.

[4]Apart from theoretical considerations, we also tested the Shazam app for several days in an Austrian dancing school with live music and out of hundreds of played songs, a single one was correctly identified, with the wrongly identified songs usually not even within the same musical genre.

[5]30s, 5s and 3s (in final system) were considered.

(Arzt and Widmer, 2010) describe a system to automatically track the position within a music piece in real-time, using online dynamic time warping. While this could in principle be used to reconstruct the temporal beat structure – and thus the dance style – from the audio data, the presented system still needs the actual score as a MIDI file to enable this alignment, which is not normally available in our case. Another advantage of such a system, could it be extended to work with arbitrary musical scores, would be to determine not only dance style but also the precise temporal position of the main beat.

(Dixon, 2007) describes a system to track beats directly from audio with a two-stage process: *tempo-induction* to find the rate of beats, and *beat tracking*, synchronizing a quasi-regular pulse sequence to the music. Both use onset detection based on spectral flux. His system has won the MIREX 2006 Audio Beat Tracking Evaluation albeit just by a small margin, but with a runtime about half as the second-placed algorithm. In *Conclusion and further work*, he notes that it would be easy to modify BeatRoot for real-time tracking. After the optimizations we have done to enable processing on platforms with less computational power, we tend to agree. Features derived from BeatRoot's output are used as one main component in the feature set for our Dancing Guide system.

(Dixon et al., 2003) describes a system for dance genre classification based on estimating periodicities (either via BeatRoot – see above – or auto-correlation), then reconstructing metrical hierarchies and estimating tempo at the beat and higher hierarchial levels. From this the dancing style is estimating using a set of simple rules. They found that – for this specific task – auto-correlation gives better results than using BeatRoot. The majority of errors is due to selecting the wrong metrical level, i.e. choosing a measure period that is double or half the correct value. As will be explained later, we prevented this issue by counting each measure period also in the bin corresponding to double and half the measure period. They quote dancing style recognition rates between 69% (using BeatRoot) and 80% (for auto-correlation data) with a much larger set of fourteen dances, however without accounting for noisy and distorted recordings from mobile phone microphones. Their classification is also done for 30s windows and not on 3s windows as in our case which probably makes the task slightly easier.

(Chew et al., 2005) introduces a method for music genre classification using a computational model for Inner Metric Analysis. They show that their method can distinguish pieces that have the same tempo and meter, however their system requires the specification of at least one template for each genre – in some cases two were needed. It is not inconceivable that for a larger set of dances the templates will have to be indefinitely extended. They also consider five genres – Tango, Rumba, Bossa Nova, Merengue, and National Anthem – but these differ from ours. A comparison of template with musical piece based on spectral weights performed more accurately than one on metric weight. An average accuracy of 70.3% is quoted. If we consider the second ranked genre as a match as well, an accuracy just below 80% is obtained. Some of the errors are due to similarities between the rhythm templates, which therefore need to be carefully designed. Again the whole piece must be analyzed before a dancing genre can be estimated.

(Gouyon et al., 2004) describe a system to classify dance music genre using rhythmic features, periodicity histogram features, inter-onset interval histogram features, and MFCC features computed on the inter-onset interval histogram. As such it is similar to (Gouyon and Dixon, 2004). They report an accuracy of 82.3% for a Nearest Neighbor classifier using ground truth tempo, however in our case this is not available. Adding the 15 MFCC-like descriptors the accuracy increases to 90.1%. Using computed instead of ground truth tempo – which is not available in our case – drops the accuracy to 78.9%. We also use MFCC features in our feature set, however those are computed from the audio spectrogram. Again the whole piece must be analyzed before a dancing genre can be estimated.

(Medhat et al., 2017) applied two neural networks specially designed for multi-dimensional audio signal recognition – CLNN and MCLNN – to the extended ballroom music dataset (which is quite similar to our dataset). They report a competitive accuracy of 92.13% without any handcrafted features, which also compares well with other deep learning approaches, but still lies below the best systems with handcrafted features. Also, these results were obtained by voting multiple predictions on each 30s samples and it is unclear whether accuracy on shorter 3s samples would be sufficient for our task.

(Yang et al., 2020) propose PRCNN, a combined convolutional and recurrent network that combines feature extraction and time-series classification, and show a slight improvement in accuracy over previous approaches on the extended ballroom dataset. Similar to our approach, they also use 3s windows. They report an accuracy of 92.5% on the extended ballroom dataset. It is implied that the reported accuracies are from the shorter 3s samples, which would indicate a major improvement. Given progress in mobile phone computing performance, it may even be feasible to run their systen in real-time on a mobile phone.

Concluding, contrary to the system presented in this paper, none of the described systems are able to run in near-real-time, except for (Wang, 2003) – which cannot process live music usefully and would also need additional information on dancing style for all indexed songs – and (Arzt and Widmer, 2010) – which needs the actual score as MIDI file, which is usually not available in our setting. Newer systems such as (Yang et al., 2020; Medhat et al., 2017) may also be able to run in near-real-time, but have been published long after our initial work. State-of-the-art performance at the start of our project was 80% accuracy which is also what we will aim for.

3 INITIAL LEARNING EXPERIMENTS

This section contains everything up to the first deployment of the model. While the preprocessing did not change after the first deployment, retraining took place and we added both feedback from known and unknown users, as well as additional data, to the training and validation datasets. The steps taken after deployment can be found in Section 4.

Tenfold cross-validation was used throughout, and reported accuracy values are at 100% coverage (i.e. without removing low-confidence predictions) unless otherwise noted. In later steps we had to resort to confidence thresholding to improve accuracy at a reduced coverage.

3.1 DATASETS

We initially chose the following thirteen dances, however later had to reduce this set to the first five dances (after merging Viennese and Slow Waltz).

1-5. Cha-Cha-Cha, Jive, Samba, Tango, (Viennese) Waltz

6-13. (Slow) Waltz, Rumba, Quickstep, Foxtrot, Polka, Quadrille, Boogie, Paso Doble

Initially we used prerecorded ballroom dance music downloaded from various webpages, but these were – although recorded in high quality – streamed in low quality. However, we soon realized that for best results we must record and process audio data in uncompressed format. So we obtained a small set of dance music CDs from small Austrian dancing schools, containing 28 dance music songs.

[6]Contains multiple deployed models.

Table 1: Unique users and feedback by year

Year	**#Users**	**#Feedback**	$\frac{\#Feedback}{\#User}$	**Accuracy by feedback**	**Coverage**	**Class unknown (1-Coverage)**	**Recording error**
2010[6]	6,031	434,812	72.10	67.29%	68.10%	31.90%	18.25%
2011	22,273	1,810,900	81.30	75.32%	62.37%	37.63%	13.53%
2012	17,851	1,434,549	80.36	72.00%	60.34%	39.66%	9.44%
2013	7,621	646,370	84.81	71.85%	61.37%	38.63%	13.82%
2014	2,239	196,671	87.84	73.15%	61.01%	38.99%	13.12%
2011-2014	**49,984**	**4,088,490**	**83.58**	**73.08%**	**61.27%**	**38.73%**	**12.48%**

For a realistic setting, we recorded the audio data directly on the phone in a room with typical ball room characteristics (parquet floor, high ceilings, irregular shaped ceiling reminiscent of chandeliers etc.) while playing the audio CD from the other side of the room. Initially samples were recorded at 44.1kHz, but later we reduced this to 8kHz to improve analysis processing speed.

For the initial dataset we used 28 dance music songs from dancing school CDs and extracted windows of length 5s from uncompressed recordings via mobile phone microphones as described above. At the very beginning we recorded a set of 30s samples, however the feedback on dancing style would have taken too long, so we switched after just a few preliminary learning experiments.

We later obtained an additional set of several dance music segments not previousy known to the system. These were recorded on a SonyEricsson mobile phone in real-life conditions and contained uncompressed audio data of 29 pieces with known dance class, each sample 30s in length, and each was added to the previously created data set. We then split this dataset into 2/3 training and 1/3 testing, making sure that there was no overlap in songs between training and testing, and extracted all non-overlapping 5s samples from each set.

3.2 PREPROCESSING

For preprocessing we initially only used the Marsyas[7] framework and only the MFCC components at various windowing sizes, since these also worked well in related work by others. However, the achievable performance was found inacceptable. So we added BeatRoot[8], an – in 2008 – state-of-the-art beat tracking system, and computed *all* reasonable features available within Marsyas – BEAT, SVLPCC and (SVSTFT)MFCC. On the observation that BeatRoot may confuse beats at double and half rates, we combined all beat estimates with their half and double beat rates, and created a vector with counts of all beat rates over the chosen window, which we called BRT. The window size was then reduced from 30s to 5s to enable a faster recognition, as waiting for half a minute was deemed too long for our use-case.

[7]https://github.com/marsyas/marsyas

[8]http://www.eecs.qmul.ac.uk/~simond/beatroot/

3.3 WRAPPER-BASED FEATURE SUBSET SELECTION

A combination of all Marsyas features and the BeatRoot vectors using a linear Support Vector Machine(Platt, 1999) (learning pairwise models for any unique set of two classes and fitting logistic models to residuals to get better confidence estimates) for training yielded 80.86% accuracy[9] which could be improved to 82.19% by optimizing the cost parameter. This was the starting point for our wrapper-based feature subset selection.

To enable faster processing, we then applied a manual feature subset selection to find the smallest subset with similar performance. First we independently removed only one related subset from the dataset and repeated the training experiments. Removing BEAT[10] features yielded an accuracy of 78.63% which was slightly worse. Removing SVLPCC yielded slightly better results at 82.48%. Removing (SVSTFT)MFCC features yielded an even worse result at 76.23%. Removing BEAT and (SVSTFT)MFCC features yielded a still worse result of 72.57%.

Removing SVLPCC and TFTM parts of SVSTFTMFCC yielded slightly better results at 82.77%. So we concluded that MFCC features performed best and used MFFC, BEAT and BRT as our new best set. Additionally removing MFCC 0-3 yielded a much worse accuracy of 76.90%, while removing MFCC 9-12 yielded 81.52% which is only slightly worse. So while MFCC bands 0-3 are more important than the higher frequencies, we left both

[9]As we mentioned, all accuracy values were obtained by ten-fold crossvalidation.

[10]These are beat-tracking features by Marsyas. The set of features by Beatroot was called *BRT*.

in. Additionally removing the low-frequency beat components of BRT (everything below 0.25 beats per minute) yielded an accuracy of 82.87% which was the best accuracy up to that point. Accordingly, it was kept as new best feature set.

After obtaining additional data and adding it to the initial training set, the BEAT features within Marsyas could be removed while still retaining an accuracy near 80% (79.26%) To enhance averaging of MFCC features, we also added min, max, range and median functions which slightly improved results. However since computing the median necessitates sorting the value vector this eventually proved to be too slow and we removed it again. Noting the significant confusion between Viennese Waltz and Slow Waltz, we merged both classes into Waltz. The first 5s segment of each song also had much higher error due to lead-in, so we removed that and the last segment for symmetry.

3.4 FIRST DEPLOYED MODEL

It later turned out that the estimates on runtime of the analysis code obtained by mobile phone emulator were far too optimistic. To obtain the needed speed on mobile phones, we switched the sampling rate from 44.1kHz to 8kHz and used 3s samples, which reduced the runtime significantly to 7s for each 3s sample. As this corresponded to 3s recording (during which processing can already start) and 4s waiting for the final result, this was deemed acceptable for near-real-time.[11] These results were confirmed on a second test phone (Sony Ericsson W959). For a test on Nokia phones, we also obtained a Nokia E50. The change in sampling rate and recording length necessitated rerecording all dance music samples on a test phone. This time we chose the Nokia E50. To enable automated recording, we used a notebook computer with active loudspeakers to play back the audio files and controlled the local recording on the phone using bluetooth, and transferred the recorded file back to the notebook directly after recording. This speeded up the local generation of training data significantly, which previously had been a manual process.

We accordingly retrained the linear SVM model using 8kHz samples with somewhat worse results, however by setting a confidence threshold at 0.5, we could still achieve an accuracy of 85.73% albeit with 58.81% of samples missing (i.e. coverage 41.19%). A parameter optimization of the SVM complexity parameter did not improve these results at this point, so we deployed it to our test phones.

[11] Note that this was measured on Nokia and SonyEricsson phones, and the first Android phone was already three times faster, so that in later versions the feedback was practically instanteneous.

However, a real-life test with the Nokia E50 yielded only 6 samples out of 18 correctly classified, while the SonyEricsson had 9 out of 18 samples correctly classified (in case of no answer the sample was repeatedly presented to the system). So we pooled samples from both mobile phones that were initially used to record the samples, manually resampling the old 44.1kHz samples and afterwards rerunning the preprocessing within the phone emulator. The idea was to pool multiple mobile phones and their corresponding microphones to get a more robust model which abstracts from concrete microphones and therefore yields more stable results. We continued with this idea during the deployment of the system towards the final model.

However, the results – while much improved – were still insufficient. Our target was having both accuracy and coverage at around 80%. By inspecting the confusion between dance classes, the number of available training samples, and after various experiments, we removed more than half of the classes, leaving the five most common: Cha-Cha-Cha, Jive, Samba, Tango and Waltz. For this set an accuracy of around 80% with 77.7% coverage could be obtained after retraining, and this is the model which we deployed for the first version.

3.5 IMPLEMENTATION DETAILS

After preprocessing, the main challenge was to port the MFCC preprocessing of Marsyas as well as Beatroot to a mobile phone. According to time and memory measurements for Beatroot and Marsyas on a mobile phone simulator for Nokia Symbian (Series 60), the initial processing time and memory was 2.48s / 29M for beatroot features (BRT), 0.078s / 12M for SVMFCC, 2.30s / 31M for BEAT features, and 0.067s / 2.6M for classification using the linear Support Vector Machine, yield 31M maximum memory consumption and a processing time of 5.09s per second of sample (i.e. five times slower than realtime) when processing 30s samples, which was acceptable for initial experiments. So we restricted the test to mobile phones with more than 32M memory (optimally with 32M free memory at system start) and with at least 360Mhz processor, which at that time gave us the Nokia N95 and the Nokia 5700.

A large part of the processing time was consumed by Beatroot, so we checked again whether the BRT features were really necessary. However, it turned out that BRT features were necessary to obtain an accuracy of at least 80% and performance drops significantly if they are removed.

By adapting the prototype code to use float instead of double we obtained the same speed but significantly smaller memory consumption. Instead of our initial estimate of 31M (obtained by analyzing 30s samples) we noted our code would work with 1M of memory (when analyzing 5s samples) and thus extend the number of mobile phones that are able to run it. It is likely that using 30s samples for the initial memory consumption test severely overestimated memory consumption.

To ensure that our audio capture code works on as many devices as possible, we obtained a test account for a mobile phone virtualization company and checked audio recording capabilities for various Nokia and SonyEricsson phones. With minor modifications we were able to run the audio recording code on all these phones. However as we already noted our runtime estimates obtained from mobile phone emulators were too optimistic by a factor of 6.3 (tested on a SonyEricsson K700i). We obtained another test phone (SonyEricsson V630i) and tested various speed improvements (such as using MathFP, an integer floating point emulation platform) but speedup was minimal. However, reducing the sampling rate from 44.1kHz to 8kHz and reducing recording time from 5s to 3s was sufficient to speed up the system to a point where it was useful again.

About this time Android phones were finally becoming available in Austria. We ported the Java code to Android and tested the preprocessing engine on the emulator. We added a more sensible user-interface and the ability to give feedback on the correctness of the results via a server-side interface on our webserver. This became available on 10th November 2009.

By porting the Java code via XMLVM[12] to Objective-C, we also created an iPhone version with reasonable effort from the same code base. This became available on the 10th March 2010.

4 EVALUATIONS DURING DEPLOYMENT

During the seven years that the system was available, we added new languages[13], fixed many bugs (as can be seen by the reduced recording error portion in Table 1 from 2010 to 2011), added small improvements, and also occasionally retrained the system using feedback and data from other sources. Since sending raw audio samples entails privacy issues, the system was designed to send only the processed vector which contains MFCC and BRT features.[14] without allowing its reconstruction. If no internet connection was available, feedback was temporarily stored and sent out once the connection became available again. This was done to ensure that no feedback was ever lost.

In January 2010 we added new dance files, bringing the total up to about 500 distinct songs, adapted the recording server to also work for Android phones, also switching from Bluetooth to Wifi to increase transfer speed for the finished recording. With this we rerecorded all new and old dance files on the Android phone using its microphone in a suitable room. This dataset had 2,694 samples. We estimated an accuracy of 80.67% at a threshold of 0.65 with a coverage of 75.84% (i.e. 24.16% of samples returned *dance class not recognized*). This updated model was deployed on 15th Jan. 2010.

In March 2010 we added a small set of training samples which were not previously used and rerecorded them using the Android test phone. Analyzing feedback from Jan-Mar 2010 only gave one user with a significant number of usable feedback, however w.r.t. this feedback our new retrained model performed worse so we did not deploy it. At this time we added test recordings from an actual Viennese ball we attended to our independent test set.[15]

We then tried to adapt the model using different threshold for each predicted class, and again used the feedback samples to evaluate this, however again the optimized model performed universally worse. We speculate that the number of songs is sufficient, however increasing the number of samples by increasing overlap could still improve results. However as this makes evaluating the system's performance much harder we did not follow it up.

As final try, we exhaustively tested about 10 million thresholds[16] extensively on the last deployed model to find a set of thresholds with coverage at least 80% (20% class unknown) and accuracy at least 80%. However it was not possible to get coverage 80% and accuracy 80% for all five dance classes. The closest was 74.71%. But since this model reduced the coverage on the actual ball data to 16.67% (i.e. 5 out of

[12]http://www.xmlvm.org/overview/

[13]Dec.2009: Spanish, French, Polish, Czech, Traditional Chinese; Jan.2010: Portugues, Dutch

[14]I.e. he vector which was used as input to the SVM which output the predicted dance class, and which was directly computed from the 3s audio samples.

[15]I.e. not the be used for training but just for validation. The main reason was that the number of samples was still deemed to be too small.

[16]0.0 to 1.0 in steps of 0.04 for each of the five classes.

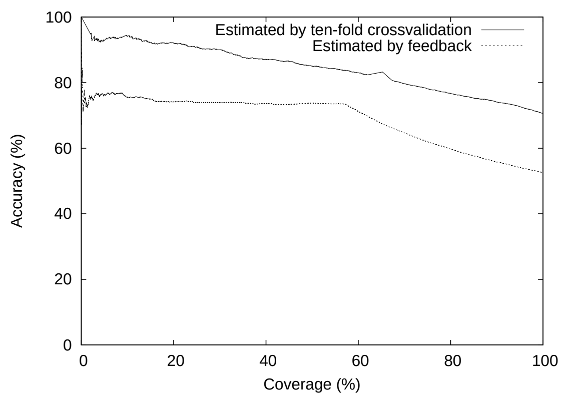

Figure 1: Coverage (X-Axis) vs. Accuracy (Y-Axis) for the final model, estimated by ten-fold CV and by user feedback.

6 samples output *dance class not recognized*), we did not use it. By optimizing accuracy, we then chose a threshold setting with a coverage of only 33.24% but an accuracy of 89.78% and put it on an iPhone, however live tests indicated that performance was much worse than the previous model, so it was again not deployed.

In June 2010 we received feedback from a test user at a dancing school for a relatively large number of dances – 100 in total with 41 correctly identified and 59 with prediction errors – and added the erroneuosly classified dances to our training set. After testing with the validation set, we found that the performance was slightly better, so we deployed the retrained system.

On 8th November 2010 we added all feedback data from June 2010 to that day to the training data and retrained the system, tested on the validation set, and deploying the new version on the 10th November. However the difference in accuracy and coverage versus the old model was quite small. This dataset had 4,780 samples.

On 13th December 2011 we collected all previous feedback samples from 8th of November to that day, yielding 6,985 Android and 4,017 iPhone samples for retraining. Retraining just on the feedbacks gave an accuracy of 63.88% for the iPhone (with threshold 0.55 at a coverage of 50%) and 63.08% for Android (with threshold 0.464 at a coverage of 50%). So – although the feedback was of a similar size as the previous training set (4,780 samples of which 1,621 were from feedback) – our old model actually performed much better than the model trained by user feedback. After investigating about 400 duplicate input vectors and assigning the majority class to all of them and repeating the analysis with the old model, very similar resolts (62.85% and 61.35%) were obtained. The old model applied to this new data gives much better results at 73.52% and 72.58%. User feedback has a 3.13x spread between most and least common class while our training data has only a 1.10x spread. This could explain the bad results when training just on feedback. However as noted in Section 5 equalizing the class distribution also does not help to get a better model. Therefore we did not deploy a new model then.

From 2011 to 2014 the deployed model was unchanged. This model had – according to estimates derived from the training set via tenfold crossvalidation – an expected coverage of 80.66% (i.e. missing 19.34%) and an accuracy of 76.47%. While the accuracy agrees well with the performance according to user feedback in Table 1 (73.08%) for these years, the coverage is much lower. This may be since we did not account for background noise in the mobile phone microphone recordings, and as such the more noisy samples fall below the confidence threshold after classification twice as often. At a comparable coverage of 59.33%, the final model would have had an expected accuracy of 83.07% – much higher than what we observed – so we can exclude a drift in confidence values as the cause. For comparison, Figure 1 shows a plot of coverage versus accuracy at all different thresholds both for the crossvalidated estimated as well as those estimated by user feedback from 2011 to 2014 where the model did not change.

Sadly, as the performance could not be significantly improved using feedback and the coverage was below our expectations, we stopped further development in 2012. The app was available on Android and Apple for several years more but was no longer maintained and unique users and feedback numbers continued to drop. We switched off the server-side interface in 2020 after having received no more feedbacks for several months.

5 DISCUSSION

It was quite disappointing that the intended regular retraining of the system using user feedback did not work well, and this contributed to the late publication of this work as well as the eventual cancellation of the project. However, one point that is obvious – perhaps only in hindsight – is that the user feedback has a much more uneven class distribution that the original data set or even the extended set used for the final model.

Therefore we revisited this issue and retrained the system using all feedback before 13th Dec. 2011 after resampling to roughly equal class distributions. We prepared a dataset with a spread of 1.28x by downsampling from the original dataset of all feedback up to 13rd Dec. 2011. We obtained an accuracy of 65.11% at a coverage of 63.85% (i.e. with 36.15% missing) by tenfold cross-validation, which is very similar to the previous result. When we apply this model which was only trained on feedback on the feedback from 2012, we get again coverage of 61.10% (38.90% missing) with an accuracy of only 49.13%. Even pooling all feedback from 2010 to 2013 and evaluating on 2014 does not improve this.

So, clearly our initial data is more useful than feedback alone even when we correct for the unequal class distribution, perhaps because it did not include noise. We therefore speculate that the non-improvement of performance indicates a limitation of the used feature set rather than lack of training data. Perhaps orders of magnitude more training data would be needed to enable higher coverage, or perhaps the signal-noise-ratio of the audio samples with background noise is simply too low.

6 CONCLUSION

We have implemented and deployed a system for dance music genre classification, which determined the correct dancing style with about 73% accuracy and a coverage of 61% (i.e. reporting an unknown class in 39% of cases). While the accuracy was as expected, the coverage was below expectations. The dancing style was reported in near-real-time – about 4s after recording a 3s sample, and later below 1s – and was therefore suitable for the intended purpose of helping novice dancing students to find the appropriate dance for a given live or recorded music.

We have deployed the system for seven years, and it was extensively used from 2011 to 2014 by about 50,000 users, each of which used it about 84 times per year (7 times per month). Feedback by users agree well with our previous estimate of the models' performance, which was most likely due to the fact that – instead of analyzing the audio samples directly – we recorded them with a variety of smartphone microphones in rooms similar to dancing halls, simulating acoustic characteristics albeit not background noise, before audio analysis. However we neglected to account for and model realistic background noise which might explain the observed lower coverage in the field.

While our system was intended for continual improvement by integrating users' feedback, this part did not work well. It may be that the initial feature set which we chose after extensive experiments was too restricted to enable a better performance, or the feedback may be of insufficient quality. The latter would however be surprising as the accuracy agrees quite well with pre-deployment estimates.

ACKNOWLEDGEMENTS

We would like to thank all volunteer translators and all others who have helped to improve our system, provided feedback, dance music or test phones, most notably Andreas H., Gregoire A., Christoph B., Ondřej J., Witono H., Melanie M., Paolo P., Martin P., Sonja S., Herbert S. and Arthur K.

REFERENCES

Arzt, A. and Widmer, G. (2010). Towards effective anytimemusic tracking. In Ågotnes, T., editor, *Proceedings of the Fifth Starting AI Researchers' Symposium*, pages 24–29.

Chew, E., Volk, A., and Lee, C.-Y. (2005). Dance music classification using inner metric analysis. In *The next wave in computing, optimization, and decision technologies*, pages 355–370. Springer.

Dixon, S. (2007). Evaluation of the audio beat tracking system beatroot. *Journal of New Music Research*, 36(1):39–50.

Dixon, S., Pampalk, E., and Widmer, G. (2003). Classification of dance music by periodicity patterns. In *Proceedings of the Fourth International Conference on Music Information Retrieval*, Baltimore (ML), USA.

Gouyon, F. and Dixon, S. (2004). Dance music classification: A tempo-based approach. In *Proceedings of the 5th International Conference on Music Information Retrieval*, Barcelona, Spain.

Gouyon, F., Dixon, S., Pampalk, E., and Widmer, G. (2004). Evaluating rhythmic descriptors for musical genre classification. In *Proceedings of the AES 25th International Conference*, pages 196–204.

Medhat, F., Chesmore, D., and Robinson, J. (2017). Automatic classification of music genre using masked conditional neural networks. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 979–984. IEEE.

Platt, J. (1999). Fast training of support vector machines using sequential minimal optimization. advances in kernel methodssupport vector learning (pp. 185–208). *AJ, MIT Press, Cambridge, MA*.

Wang, A. (2003). An industrial-strength audio search algorithm. In Choudhury, T. and Manus, S., editors, *ISMIR 2003, 4th Symposium Conference on Music Information Retrieval*, pages 7–13.

Yang, R., Feng, L., Wang, H., Yao, J., and Luo, S. (2020). Parallel recurrent convolutional neural networks based music genre classification method for mobile devices. *IEEE Access*, Preprint.