Applying Learning Classifier Systems to Acoustic Scene Classification: DCASE 2017 Challenge

CITS4404 Artificial Intelligence & Adaptive Systems Team Project

Yiyang Gao (21263128), Aaron Hurst (21325887), Kevin Kuek (21307006), and Scott McCormack (21875529)

School of Computer Science and Software Engineering

3rd November, 2017

Abstract

Sound classification is an emerging field of research with multifarious technology and big-data applications. To promote research in this space, the DCASE Challenge was instigated with multiple tasks for competitors to attempt. Here, we have investigated the efficacy of Learning Classifier Systems applied to the problem of Acoustic Scene Classification. This involves predicting the environment in which a sound file was captured. Results indicate that even on a highly abbreviated feature set, Learning Classifier Systems can achieve modest accuracy in comparison to standard solutions.

1 Introduction

Write introduction

2 Background

This section provides a brief review of learning classifier systems (2.1), the DCASE Challenge (2.2) and acoustic scene classification (2.3).

2.1 Learning Classifier Systems

First introduced in the mid-1970s, Learning Classifier Systems (LCSs) are a rule-based machine leaning algorithm with a unique combination of learning mechanisms, including a genetic algorithm (GA) [1]. The core of an LCS is a population of rules, or *classifiers*, which collectively form the solution to the given problem [2]. This population of classifiers is gradually evolved toward an optimal and maximally general set [2].

The motivation for this structure is that, when modelling and attempting to predict the outcome of complex systems, a desirable approach is to develop a distributed population of classifiers – in the form of rules – that together form an accurate model [2, p. 2]. Each classifier, then, spans a subspace of the problem, with the population spanning the entire problem. Individual classifiers consist of a condition-action rule which says: 'If a problem instance matches this *condition*, perform this *action*'. Classifier fitness is evaluated based on feedback from the problem (generally referred to as the 'environment').

The learning process of a LCS includes a rule discovery method known as covering, a generalisation-pressure effect called subsumption, a GA, fitness-based deletion to maintain a finite-sized population and, in some applications, reinforcement learning [3].

The rule discovery method, covering, is used to initialise the population by adding a new classifier whenever a problem instance matches no existing classifier. Subsumption is used to eliminate classifiers with more specific conditions that are no more accurate than others with equivalent, but more general conditions. The GA is used as a secondary rule discovery method that only operates on high performing rules. Deletion is employed to maintain a finite population size by removing poorly performing – i.e. low fitness – classifiers. Reinforcement learning may be used as a final step in the learning cycle for problems where feedback from the environment is delayed.

A seminal work in the field of LCSs was the introduction of the eXtended Classifier System [4, 5]. This incorporated a number of features which substantially improved the performance of LCSs [4].

A key distinction amongst LCS applications is between supervised learning, in which feedback from the environment is delayed (such as robot navigation), and offline learning, where feedback is immediate and the correct action known in advance (such as classification tasks). This distinction determines whether reinforcement learning is necessary and affects how classifier accuracy is calculated.

2.2 DCASE Challenge

Sound classification, or machine listening, is seen as a promising research field with wide-ranging applications [6]. The DCASE challenge has been established to encourage and support work in this space. The challenge provided participants with standard development (training) and evaluation (testing) datasets [7] and a baseline system for comparison and/or extension [6, 8].

The Challenge spans four sound recognition tasks: acoustic scene classification, detection of rare sound events, sound event detection in real life audio and large-scale weakly supervised sound event detection [6]. This paper focuses on acoustic scene classification. For this task, the Challenge provides datasets containing sound files obtained across 15 different contexts, such as in a car, library or office [8]. Each sound file in 3-5 minutes long natively, but was split into multiple 10 second long segments for the datasets [8]. Overall, 312 segments are provides for each context.

A baseline system for this task is provided in Python and uses a neural network with two hidden layers of 50 neurons each to classify sound files [6]. The baseline system also provides support for extracting features from the datasets using what is known as log mel-band energies, as described in Section Section 3.2 [6].

2.3 Acoustic Scene Classification

The first task of the 2017 DCASE Challenge, and the focus of this paper, is acoustic scene classification. Barchiesi et al. define this as "the task of associating a semantic label to an audio stream that identifies the environment in which it has been produced" [9, p. 17]. Potential applications in this domain revolve around context aware smart devices, such as smartphones and hearing aids, that adjust their functioning based on the environments in which they find themselves [9].

The typical approach taken to acoustic scene classification is to segment the original sound file into many, small 'frames', calculate a set of features over each frame, use the resulting feature array to train a statistical model and finally apply some decision rule for assigning classification labels

[9, pp. 18–19]. Many approaches submitted to the 2017 DCASE challenge trained some form of neural network as their 'statistical model' [8].

The results of the 2017 DCASE Challenge show that the baseline solution provided achieved an accuracy of 74.8% on the development dataset and 61.0% on the evaluation set averaged across all sound classifications [8]. Many entrants successfully outperformed the baseline on both the development and evaluation datasets; however, all algorithms performed worse on the evaluation set compared to the development set [8]. The best performing solution achieved an accuracy of 87.1% and 83.3% on the development and evaluation datasets, respectively [8, 10].

3 Feature Extraction

3.1 Feature Engineering

Feature engineering is a vital aspect of machine learning, as the way that data is presented to a predictive model has a large impact on the quality of results achieved [11]. This is because features that dont capture important information from the dataset effectively will not provide an accurate representation of the environment and may ignore useful patterns. Feature extraction is a method of deriving key information from a dataset into a useful format by using tools specific to the problem domain and typically results in a reduction in data dimensionality [12]. The Curse of Dimensionality is an important concept in the domain of machine learning that refers to the problem that as the dimensions of feature set increases, the volume of the feature space increases exponentially [13]. This has a particularly large impact on LCSs, as, with the exception of wildcards, the rule for every single feature is required to match with that of the instance to classify. To illustrate this, for a feature set of size 1000 and a classifier with 50% wildcards, still 500 features from the environment instance would need to perfectly match the classifiers corresponding rules and if even one of them does not, the instance will not be matched. One way to interpret this is that for a highly dimensional feature set each individual feature is insignificant [13], but can have a large impact on the result.

For the application of acoustic scene classification, defining features need to be extracted from sound files, and these need to be reduced to a dimensionality suitable for use in LCS rules.

3.2 Feature Extraction

The mel scale is a scale of pitches that was designed to vary linearly with a listeners perception of a sound and is used in the LCS for sound data representation. It is an informal unit of measure that can be converted to from the frequency spectrum by using a function derived from what listeners judge to be pitches of equal distance from each other [14]. There are various ways of converting from Hz to mels and a common method uses the equation [15], a plot of which can be seen in

reference hz2mel.png

.

Acoustic classifiers often convert frequencies to the mel scale because its characteristic of mimicking human perception, and due to its logarithmic transformation of the frequency spectrum, it also results in dimension reduction when using bands as features [16].

— hz2mel here —

Initial feature extraction was done using the DCASE Baseline systems included feature extractor, which makes use of python music and audio analysis package called LibROSA [17]. The feature extractor takes an audio file as an input and outputs an array in which each row is a time slice

Table 1: DCASE Baseline feature extractor parameters used

Parameter	**Value**
Maximum frequency for calculating MEL band	22,050
Minimum frequency for calculating MEL band	0
Sample frequency	44,100
Hop length (samples)	882
Hop length (seconds)	0.02
Use htk style mel conversion	false
Use log scale	true
Feature extraction method	MEL
Average multichannel audio to single channel	true
FFT length	2,048
Number of MEL bands	40
Normalise MEL bands	false
Type of spectrogram	magnitude
Window length (samples)	1764
Window length (seconds)	0.04
Window type	Hamming asymmetric

and each column is a feature, with the specifics dependent on a configurable parameters list. The parameters used can be seen in

reference parameters figure

.

Based on the settings used, the feature extractor output arrays with 501 time slices and 40 features, with each feature being the log of the magnitude of a band on the mel scale. This results in a feature set for each sound file of dimensions.

3.3 Feature Reduction

The feature set extracted using the baseline system has a very high dimension, and in that form would be unsuitable for use with an LCS, as a feature set of that size would cause increased processing time as well as classifiers that are overfitted to specific instances. In order to perform dimensionality reduction without losing too much important information about that datasets, some visualisations were made to allow visual inspection and identification of trends to potentially be exploited. Each of the graphs shown in

reference beachgrid.png, cargrid.png, officegrid.png

show the features extracted for a given sound file, with 9 different sound files of a given classification displayed together. Each line represents one of the 501 time slices and the magnitude of each mel band in that slice.

—3 grids here in 1 figure—

Comparing these graphs, it is clear that there are patterns common to individual classifications and that in a given sound file, the distribution of mel band magnitudes follows a similar pattern for many of the time slices. Given this similarity between time slices, it was decided that the mean and standard deviation of each mel band magnitude across all time slices would be used as the feature set, as this reduces the dimensionality by 250 times while maintaining a measure of both the trend and spread of the data. The three example classes shown in

reference beachgrid.png, cargrid.png, officegrid.png

are shown again in

reference beach.png, car.png, office.png

, with the difference that each line represents the average log magnitude of for each mel band in one sound file of the class shown rather than each line being one time slice of a single file. These graphs illustrate that by taking the mean magnitude for each mel band, the general trends are preserved, and though the spreading of these values between time slices is not represented, it is captured in the standard deviations for each mel band. The average mean and standard per mel band across all sound files of each example classification is shown in

reference 3means.png, 3stds.png

and illustrates that the differences between classifications is still captured by the chosen features despite the dimension reduction.

—3 graphs (beach car and office.png) here in 1 fig—

—2 graphs (3means and 3stds (lol)) in 1 fig—

The amount of spread between different instances of each class observed in

reference beach.png, car.png, office.png

is noticeably large and would result in classifiers needing to have a large range of accepted values in order to catch many other instances of their classification. This spreading is potentially due to the sound files having different recorded volumes which may result in magnitude offset observed. In attempt to rectify this, the means were normalised such that in a given sound file, instead of using the means as features, the average across the means of each mel band was taken and the features used were the mel band means subtract the average mean. This method would make magnitude offset irrelevant while maintaining trends and is illustrated in

reference normalise.png

with the results displayed in

reference beachVar.png, carVar.png, officeVar.png

confirming that the general trends are preserved while reducing spread on the y axis. Unfortunately, upon testing this method was found to reduce classification accuracy so it was not used for further testing. Further analysis of the feature set showed that this is likely due to the offset of the mel band magnitudes being a defining characteristic of different scenes, which can be observed in

reference allMeans.png, allMeanVars.png

in which the average means of each class are plotted on one graph and the average normalised means are plotted on another. The graph of average means shows that classes are distinct from one another based on their magnitude offset and the other shows that by normalising them this feature is lost. The reduction in accuracy may imply that either the difference in trend was not enough to make classes distinct from one another or potentially that more parameter tuning of the LCS would be needed to take advantage of the normalisation.

—normalise.png here—

—beachVar, carVar and officeVar.png here—

—allMeans and allMeanVars.png here—

4 Experiment 1: Baseline System

Preliminary investigation into LCSs built by others, led us to sourcing from a sample system hosted on GitHub titled *eLCS*. This Michigan-style classifier was developed as an educational tool which assisted with demonstrating the intricacies and components that contribute towards building a typical LCS. Five demo folders are utilized to show the steps towards building a typical LCS and a dataset consisting of discrete attributes and a starting rule population is provided to run the LCS. The last folder in this system, titled *Demo 5* was utilized as a starting point for our baseline system.

5 Experiment 2: Custom LCS Implementation

In response to the positive results obtained from the baseline system described above, a separate LCS algorithm was coded by the authors. This provided much greater ability to control various aspects of the algorithm and more clarity as to the overall system's functioning. This section presents the general approach to developing this algorithm (Section 5.1), a description of each of the key parameters (Section 5.2) and the key modifications made to the standard LCS design (Section 5.3).

5.1 Overview of Approach

The LCS algorithm developed by the authors was closely based on the structure and pseudocode provided by Butz and Wilson in their paper entitled "An Algorithmic Description of LCS" [3]. This paper provides detailed descriptions for each of the key modules of XCS, which is one of the most well-regarded LCS implementations [5]. The decision to embark on developing a custom LCS implementation was largely based on the availability of the pseudocode in this paper.

Two fundamental changes made to the algorithm presented by Butz and Wilson were the change to continuous data, and hence continuously-valued classifier rules, and the switch from reinforcement learning to offline learning. Both changes necessitated a significant number of alterations throughout the algorithm. Various papers were consulted for how to make these changes. In particular, a number of papers on XCSR – the standard continuous data version of XCS – were referenced [18, 19, 20, 21].

Continuous valued rules were implemented using the centre-range encoding [18]. In this scheme, each rule within a classifier's condition matches values between centre - range and centre + range. Note that this representation, and indeed any continuous representation, doubles the number of "alleles" in a classifier with two for each rule [18].

Additionally, the GA required a number of design decisions. These included the use of two-point crossover, roulette-wheel parent selection and employing GA subsumption (where parents can subsume children if they are more general). The GA was also applied in a 'niched' manner to act only on the correct set. Niched GA operation has been noted as a significant contributor to LCS performance [4]. Accuracy-based fitness, another key performance enhancing feature [4], was also used.

As discussed in Section 3.3, the mean and standard deviation of log mel-bands were used as the features for each sound file in this experiment. All 312 sound files from all of the 15 sound contexts were considered. That is, the LCS *environment* consisted of 312 *instances* (sound files) per *endpoint* (context), giving a total of 4,680 instances, each containing 80 *attributes* (features; 40 means, 40 standard deviations). Data was split into training and testing datasets with a 60:40 ratio.

5.2 Parameters

Import Yiyang's section

5.3 Problem-Specific Modifications

Four components of the custom LCS implemented here required appreciable modification from standard practice. First, in designing the mutation scheme for the continuously-valued rules, it was desired to allow for each possible case. That is, mutation from specific values to wildcards, wildcards to specific values and specific values to other specific values. The methods used for each case are listed in Table 2. Note mutation may occur independently on any rule centre *or* range value independently, with the exception that a given centre-range pain must either both be specific values or both be wildcards; never can one be a wildcard and the other not.

Table 2: Mutation methods

Mutation type	**Mutation method**
Specific value to wildcard	Substitute associated centre *and* range with wildcards ('#')
Wildcard to specific value	Set associated centre *and* range to value from current instance from environment plus a random, randomly signed increment
Specific value to specific value	Add a random, randomly signed increment

Table 3 lists the probabilities of various quantities of alleles being mutated in a given classifier based on a mutation probability of 0.02, which is relatively standard [3]. As shown, there is a slightly less than 4% chance that mutation will not affect a given child. Combined with the probability of covering being set to 0.7 (also within the range of standard values [3]), this means that there is a 2.7% chance of a given child classifier being identical to its parent (in which case it will obviously be subsumed).

Table 3: Mutation probabilities

Quantity	**Probability**
0	0.04
1	0.13
2	0.21
3	0.24
4	0.18
5	0.11
≥ 6	0.10

Second, as per standard practice, correct set subsumption was used in this implementation; however, a different metric for determining the most general classifier in the correct set wa required

[3]. In XCS, the metric is to simply choose the classifier with the greatest number of wildcards. In this implementation, and additional criteria was added so that if more than one classifier has the equal most wildcards, the classifier with the largest total range is selected as the most general. This does not appear to be standard practice, but does fit intuitively.

Third, the conditions for one classifier to subsume another were drastically relaxed. This was motivated by the fact that subsumption, which is critical to evolving maximally general classifiers, was not occurring. The explanation for this is simply that it is extremely unlikely.

Consider the standard criteria for a classifier to be "more general" than another. To do so, its rule intervals must cover that of the classifier it is proposing to subsume for *all* attributes and, if the proposed subsumee has a wildcard at *any* attribute, the proposed subsumer must also have a matching wildcard [18, 20].

With the parameters used in this LCS implementation, the probability of a given rule being a wildcard is approximately 0.15. Hence, the probability of a proposed subsumee having a wildcard where the subsumer does not at any given attribute is 0.15 x 0.85 = 0.128. Across 80 rules, this means that there there is only a 1 in 55,000 chance that a subsumption match is possible. However, this analysis does not even take into account the criteria for the subsumer to have superior ranges than its subsumee, so this is actually an upper bound on the true probability.

To remedy this, the criteria for the subsumer to match each wildcard in the subsumee has been removed and a tolerance has been added to the interval matching criteria. That is, a classifier can still be said to be more general than another if its rule interval, when expanded by some tolerance factor, covers the other rule. Overall, this represents a weak form of subsumer generality that does not appear to be used in the literature. However, the authors would suggest that modifications of this nature are essential to ensuring subsumption can occur in problems with large numbers of attributes and continuous data.

Fourth, the experience threshold for deletion of a classifier from the population has been made dependent on the classifier's age. This was motivated by the observation that many classifiers require many iterations to reach the standard threshold for candidacy for deletion (a match count of at least 20). As a result, as soon as any classifiers do reach it they are very likely to be deleted since they are the only classifiers eligible for deletion. However, this is undesired as these are often reasonably well-performing classifiers.

Consequently, the authors devised an alternative deletion experience threshold that rolls off after a specified age (number of iterations since the classifier's birth). The rule for this threshold is defined by the following equation and illustrated in Figure 1

$$\frac{\text{exp}_{min}}{3.05}\left(\pi/2 - \arctan\left(\frac{\text{age} - \text{age}_{min}}{20}\right)\right)$$

Where exp_{min} is the initial minimum experience threshold (that will apply to new classifiers), age is the number of iterations since the classifier's birth and age_{min} is the threshold age after which the experience threshold drops off substantially.

6 Results

Yiyang: feel free to mess around with the sub-headings under Results if you wish

6.1 Environment Representations

Alternative feature processing investigated

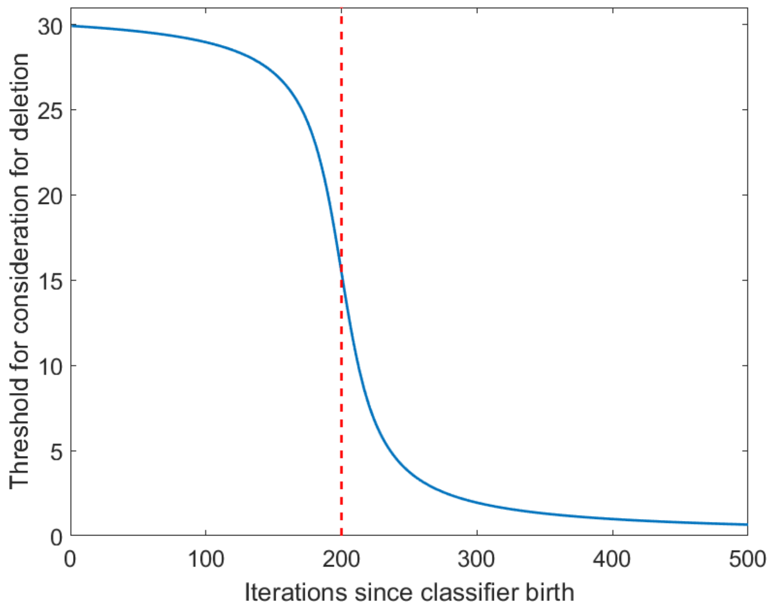

Figure 1: Variable deletion threshold with cut-off age depicted as dashed vertical red line

6.2 Parameter Tuning

Reason why standard deviation and mean are used, what information they carry

Car and office confusion matrix Plot for std dev and mean (for car and office) to explain Car, office and city centre (with confusion and plots) Then add metro station and show results (worse) - explain why it is worse Show a classifier on a graph to show how it can match instances Overall accuracy (all 15 features)

7 Discussion

Our approach is good because it is more general (no need to construct statistical models or decision rules) and significantly reduces the number of features that need to be analysed, and therefore the complexity of the algorithm. [THIS IS COMPARING TO STANDARD APPROACH TO ACS / DCASE CHALLENGE SUBMISSIONS]

8 Conclusion

Write conclusion

References

[1] M. V. Butz, "Learning classifier systems," in *Springer Handbook of Computational Intelligence*. Springer, 2015, pp. 961–981.

[2] R. J. Urbanowicz and J. H. Moore, "Learning Classifier Systems: A Complete Introduction, Review, and Roadmap," *Journal of Artificial Evolution and Applications*, vol. 2009, pp. 1–25, 2009.

[3] M. V. Butz and S. W. Wilson, "An Algorithmic Description of XCS," in *International Workshop on Learning Classifier Systems*. Springer, 2000, pp. 253–272.

[4] P. L. Lanzi, "Learning Classifier Systems: Then and Now," *Evolutionary Intelligence*, vol. 1, no. 1, pp. 63–82, 2008.

[5] O. Sigaud and S. W. Wilson, "Learning Classifier Systems: A Survey," *Soft Computing*, vol. 11, no. 11, pp. 1065–1078, 2007.

[6] Mesaros, Annamaria and Heittola, Toni and Dimint, Aleksandr and Elizalde, Benjamin and Shah, Ankit and Vincent, Emmanuel and Raj, Bhiksha and Virtanen, Tuomas, "DCASE 2017 challenge setup: Tasks, datasets and baseline system," in *Detection and Classification of Acoustic Scenes and Events*, 2017.

[7] Mesaros, Annamaria and Heittola, Toni and Virtanen, Tuomas, "TUT Database for Acoustic Scene Classification and Sound Event Detection," in *24th European Signal Processing Conference 2016 (EUSIPCO 2016)*, 2016.

[8] A. Mesaros and T. Heittola, "Acoustic scene classification - dcase2017," 2017. [Online]. Available: http://www.cs.tut.fi/sgn/arg/dcase2017/challenge/task-acoustic-scene-classification

[9] D. Barchiesi, D. Giannoulis, D. Stowell, and M. D. Plumbley, "Acoustic scene classification: Classifying environments from the sounds they produce," *IEEE Signal Processing Magazine*, vol. 32, no. 3, pp. 16–34, 2015.

[10] S. Mun, S. Park, D. Han, and H. Ko, "Generative adversarial network based acoustic scene training set augmentation and selection using SVM hyper-plane," DCASE2017 Challenge, Tech. Rep., 2017.

[11] J. Brownlee. (2014) How to Engineer Features and How to Get Good at It. [Online]. Available: https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/

[12] J. Howbert. (2012) Machine Learning Feature Creation and Selection. [Online]. Available: https://courses.washington.edu/css490/2012.Winter/lecture_slides/05a_feature_creation_selection.pdf

[13] E. Keogh and M. Abdullah, "Curse of dimensionality," 2010.

[14] O. Luening, A. W. Slawson, G. Ciamaga, J. Chadabe, J. E. Rogers, G. Mummam, and J. H. Appleton, *The Development and Practice of Electronic Music*. Prentice-Hall, 1975.

[15] D. O'Shaughnessy, *Speech communication: human and machine*. Addison-Wesley, 1987.

[16] D. Stowell and M. D. Plumbley, "Automatic large-scale classification of bird sounds is strongly improved by unsupervised feature learning," 2014.

[17] T. Heittola, A. Diment, and A. Mesaros. (2017) Machine Learning Feature Creation and Selection. [Online]. Available: https://tut-arg.github.io/DCASE2017-baseline-system/index.html

[18] D. Sowden, "Investigating the Learning Classifier Systems XCSR and XCSF," 2007. [Online]. Available: http://www.cs.bath.ac.uk/~mdv/courses/CM30082/projects.bho/2006-7/Sowden-DJ-dissertation-2006-7.pdf

[19] C. Stone and L. Bull, "For real! XCS with continuous-valued inputs," *Evolutionary Computation*, vol. 11, no. 3, pp. 299–336, 2003.

[20] S. W. Wilson, "Get Real! XCS with Continuous-Valued Inputs," in *Learning Classifier Systems: From Foundations to Applications.* Springer-Verlag, 2000, pp. 209–219.

[21] M. Behdad, L. Barone, T. French, and M. Bennamoun, "On xcsr for electronic fraud detection," *Evolutionary Intelligence*, vol. 5, no. 2, pp. 139–150, 2012.