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 to allow competitors to apply audio processing techniques to solve different types of audio classifcation problems. In this report, 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

The Detection and Classification of Acoustic Scenes and Events (DCASE) challenge is an annual competition which sets a number of sound related tasks for competitors to complete. The event encourages competitors to utilize past solutions, allowing submissions to regularly push the boundaries of machine sound perception and continuously improve on previously built models. The vast majority of entrants to the competition use neural networks to train models, so the application of a Learning Classifier System (LCS) to this problem was seen as a great alternative to exploring its use in application of sound classification.

LCSs function by maintaining a population of classifiers that is continuously updated using various methods. These include the application of a Genetic Algorithm (GA) to breed classifiers with high fitness and a subsumption mechanism to pressure them into being more general. The population of classifiers is the predictive model that LCSs produce and when testing where an instance is classified based on a vote from the population.

Two primary experiments were undertaken to test the appropriateness of LCSs for the DCASE acoustic scene classification task. Both of these approaches made used of a feature set created by reducing the dimensionality of the features output by the DCASE baseline system's feature extraction tool, with the first experimenting with a pre-built LCS and the second building an LCS from the ground up. Various experiments were performed on both systems to optimise the LCSs' parameters and determine the how well these systems are able to classify scenes based on acoustic features.

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*'. In these condition-action pairs, each condition is actually a set of conditions corresponding to the features in the instance's feature set and may either contain a value to be matched, or a "wildcard" that can match any value. In order for a condition to match a problem instance, every feature in the instance must matched its corresponding condition in the classifier. 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 method 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 (XCS) [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 of their own systems [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 don't 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 classifier's 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 high 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 (sound frequencies) that was designed to vary linearly with a listener's perception of a sound. It is used in the DCASE Challenge and the LCS used in this project for sound data representation. It is an informal unit of measure that can be converted to and 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 Figure 1.

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

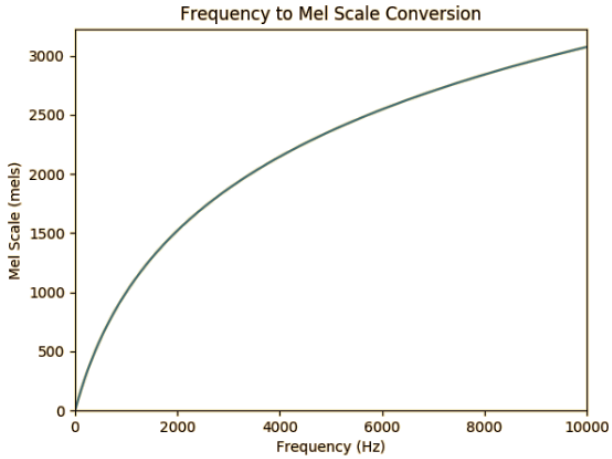


Figure 1: Conversion from frequency to mel scale

Initial feature extraction was done using the DCASE challenge baseline system's included feature extractor, which makes use of the 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 and each column is a feature, with the specifics dependent on a configurable parameters list. The parameters used can be seen in Table 1.

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 20,040 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 the datasets, some visualisations

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

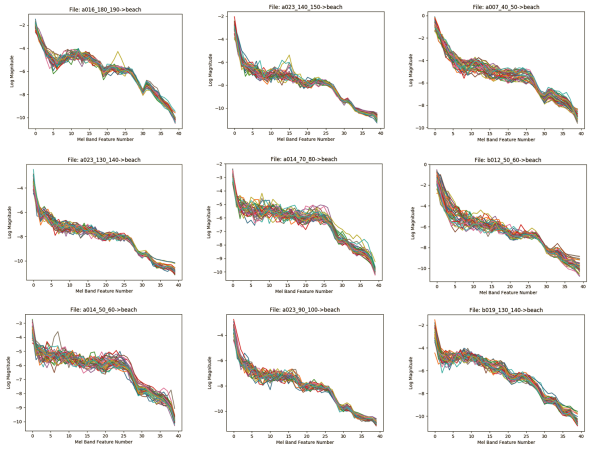
were made to facilitate identification of trends to potentially be exploited. Each of the graphs shown in Figure 2 show the features extracted for a given sound file, with 9 different sound files of a given classification displayed together for the beach, car and office classifications. Each line represents one of the 501 time slices and the magnitude of each mel band in that slice.

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 a factor of 250 while maintaining a measure of both the trend and spread of the data.

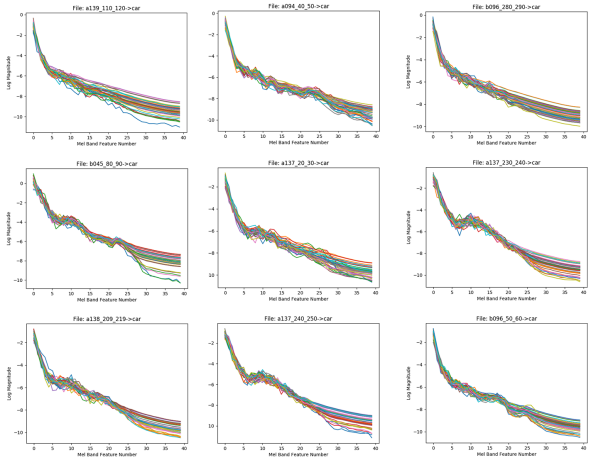
The three example classes shown in Figure 2 are shown again in Figure 3 with the difference that each line represents the *average* log magnitude of for each mel band in one sound file of the given class, rather than each line being one time slice of a single file. These graphs illustrate that by taking 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 Figure 4 and illustrates that the differences between classifications is still captured by the chosen features despite the extreme dimension reduction.

The amount of spread between different instances of each class observed in Figure 3 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 the magnitude offsets observed.

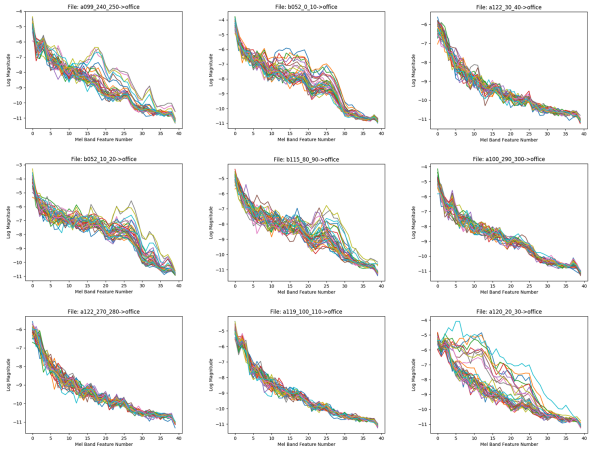
In attempting to rectify this, the means were normalised such that in a given sound file, instead of using the means as features, the average across the means of each mel band was taken and the features used were the mel band means subtract the average mean. This method would make



(a) Beach

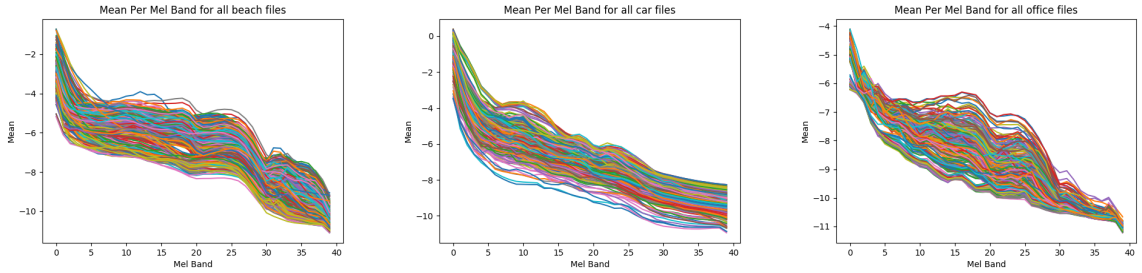


(b) Car



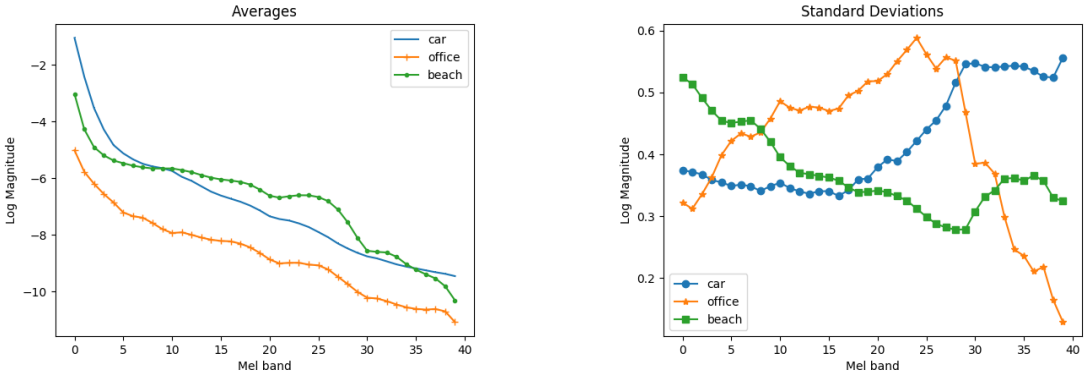
(c) Office

Figure 2: Extracted features for beach, car and office sound files



(a) Beach (b) Car (c) Office

Figure 3: Average mel bands for each sound file in beach, car and office classifications



(a) Means (b) Standard deviations

Figure 4: Average and standard deviation of sound file averaged mel bands across all sound files for a given classification

magnitude offset irrelevant while maintaining trends and is illustrated in Figure 5 with the results of this transformation displayed in Figure 6 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 discontinued. 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 Figure 7 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 discrimination 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.

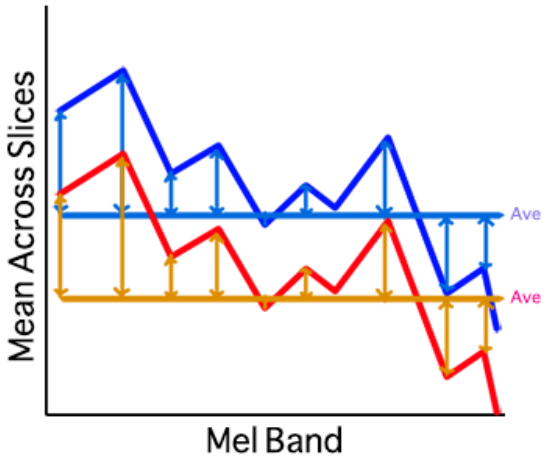
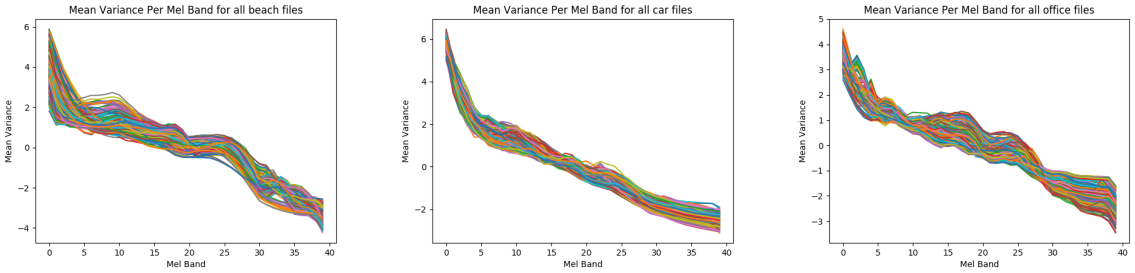
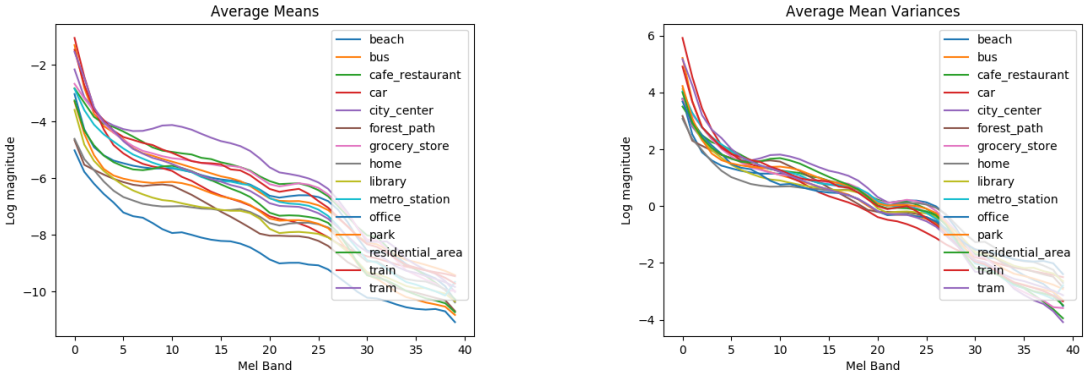


Figure 5: Normalisation procedure for feature data



(a) Beach (b) Car (c) Office

Figure 6: Normalised mean mel band values for each sound file in each classification shown



(a) Means (b) Standard deviations

Figure 7: Comparison of original and normalised mel band averages across each classification

4 Experiment 1: Baseline System

An investigation into LCSs built by others led us to sourcing from a sample system hosted on GitHub titled eLCS [18]. This Michigan-style classifier created by R. J. Urbanowicz was developed as an educational tool that demonstrated the intricacies and components that contribute towards building a typical LCS. It consists of five demo folders showing the steps towards building a typical LCS and a dataset consisting of discrete attributes and a starting rule population to run the LCS. For our purposes, we used the last folder 'Demo 5' as the starting point for constructing our LCS. This was used for the initialisation of the Github repository used for this experiment [19].

Preliminary steps towards understanding the functional processes of this LCS included refactoring and inserting comments while stepping through the code using a debugging process. From this it was discovered early in the process that the LCS had support for continuous values for attributes and could be initialised without a predefined rule population. These features which were not apparent during our project introduction presentation, greatly simplified the process of adapting our dataset to work with this implementation of an LCS. Other results from these works included: refactoring the LCS code to a submodule, creation of a 'utils' directory for preprocessing of feature files, creation of a 'notebooks' directory for visualising the results and a 'docs' directory for holding compiled documentation files. PDF generated documentation can be found at the root level of the repository.

In order to utilize the dataset processed by the libROSA library in the DCASE baseline solution, further processing of this dataset needed to take place to make it suitable for our implementation. As the dataset was too large for importation into our LCS model, the first approach was to reduce its dimensionality by calculating the mean and standard deviation over the time series data. When calculated over the entire range, it reduced the size of the dataset by a factor of 256:1. Other approaches included dividing the time series data into even slices and calculating the mean and standard deviation on these individual segments. After applying this process the dataset was then optionally split into training and testing datasets of varying ratios, ensuring that an even number of features were maintained in each dataset.

Table 2: LCS Baseline models using training data only

Model	**Segments**	**Population size**	**Maximum iterations**
1	1	1,000	500,000
2	8	1,000	500,000
3	1	3,000	30,000
4	1	5,000	50,000

For the first series of simulations the entire training dataset (4680 instances) was used to generate the rule population for feature prediction without any testing data. Although this results in a grossly overfit prediction model, it seeks to determine the performance of the LCS to generate rule populations that are able to correctly classify elements in the training dataset. The models generated in this section are listed in Table 2.

Figure 8 shows a comparison of Models 1 and 2 in which we evaluate the rule population accuracy with an increase in the number of segments to the time series data. These simulations were also run for a large number of iterations to evaluate how accuracy changed with time. A comparison of these models has shown that a strong correlation between segmentation size and population rule accuracy. It also showed that running the model for an extended period of time

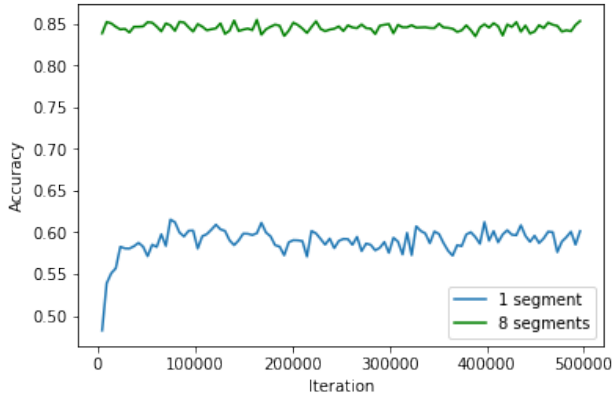


Figure 8: Comparison of LCS performance for Model 1 and 2

had no bearing on increasing its accuracy. Model 1 was shown to have benefit from running up until approximately 100,000 iterations, while Model 2 had no benefit at all from running many iterations.

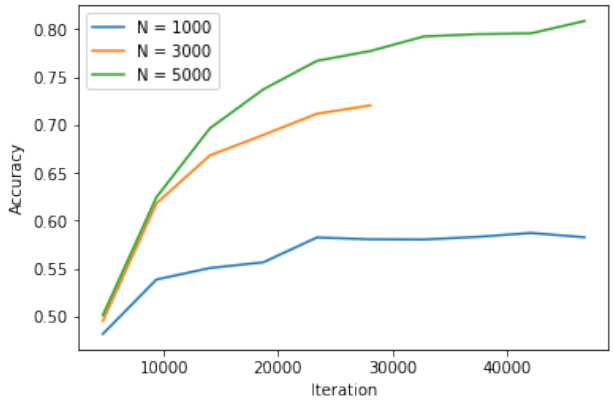


Figure 9: Comparison of LCS performance with different population sizes

Figure 9 shows a comparison of Models 1, 3 and 4 in which we evaluate the rule population accuracy with an increase to the population size up to a maximum of 50,000 iterations. These results showed that a strong correlation existed between population size and population rule accuracy. It should be noted however that increasing the population size also resulted in a substantial increase to computation time.

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 [20, 21, 22, 23].

Continuous valued rules were implemented using the centre-range encoding [20]. 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 [20].

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

The behaviour of LCS is controlled by a total of 14 parameters. These parameters are in direct relationship to the overall accuracy of the system. The list below provides a brief explanation of the function of each of the parameters.

1. **Maximum Number of Iteration**: This number determines how many iteration the LCS system will run through the data set. The system will stop once its iteration count has reached this number.

2. **Maximum Population Size**: This number controls the size of the number of classifiers allowed to reside in the population. Once the numbers of classifiers discovered by LCS system reached this threshold, deletion will occur. A low value on this parameters results in frequent deletion, which in turn allows more general rules to survive.
3. **Covering Wildcard Probability**: This parameter applies to the covering stage of the learning process. It specifies the probability of each attribute of a given instance being casted into a wildcard, meaning that this newly generated rule will disregard the value of this attribute in its decision making. A large number on this parameters tends to generate more general rules, as it is more prone to ignore an attributes value in deciding on an action.
4. **Initial Range Factor**: This parameter also controls the behaviours of LCS in covering stage. As discussed in the previous section. If an attribute is not casted into a wildcard, then the rule generated through covering will set the centre of this attribute to be the current value, and its range will be determined by multiplying its current value with the initial range factor. A large value will generated rules covering a large spectrum of the dataset.
5. **Power Parameter**: This value is used to convert a classifier's accuracy to fitness, which is needed in the GA subsystem. Fitness of a classifier is calculated using the following formula: $Fitness = (Accuracy)^{PowerParameter}$. This value is typical set as 5. Other values were also investigated to observe the effect.
6. **Deletion Threshold**: A classifier is considered as a potential candidate for deletion when its experience i.e the number of times this classifier has been matched by an instance is above this threshold.
7. **Deletion Fitness Scale**: A classifier is also considered as a potential candidate for deletion when its fitness it below a percentage of the average fitness of the entire population. This percentage is controlled by this deletion fitness scale parameter.
8. **GA Threshold**: This parameters controls the frequency of GA being run among the correct set of classifiers. The GA subsystem is only allowed to run when the average of iteration since last GA of all classifiers exceeds this threshold. A small number on this parameter will result in GA operating more frequently, will in turns leads to more rules being discovered and added in the population.
9. **Probability Crossover**: The parameter specifies the probability a child will be generated from a crossover process in the GA subsystem.
10. **Probability Allele Mutation**: The probability an attribute of a classifier undergo allele mutation.
11. **Probability Wildcard Mutation**: The probability an attribute of a classifier get casted into a wildcard.
12. **Mutation Scale**: This extent to which the centre and range of a classifier can be mutated.
13. **Subsume Exp Threshold**: The threshold the experience of a classifier need to exceed before it is allowed to subsume another classifier.
14. **Subsume Accuracy Threshold**: The threshold the accuracy of a classifier need to exceed before it is allowed to subsume another classifier.

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 3. 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 3: 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 4 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 4: 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 [20, 22].

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 10

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

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

This section reports on the results of Experiment 2. Section 6.1 discusses the results of varying some of the LCS parameters discussed previously. Section 6.2 describes the investigation of classification on subsets of the set of classes in the dataset.

6.1 Parameter Tuning

The parameters of LCS has a significant impact upon the performance of the system. Initially a separate GA component specifically for parameter tuning was proposed, to explore different combination of parameters that can produce optimum accuracy for the LCS system. However, it was soon realized that it is infeasible to do so as under current implementation each run of the LCS takes around 15 to 18 minutes to finish, meaning that GA would not be able to explore enough search space of the parameters given the limited time available.

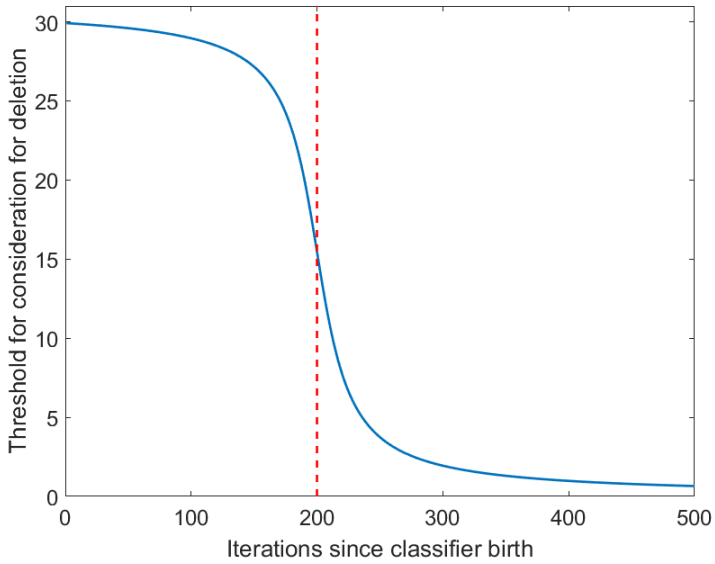


Figure 10: Variable deletion threshold with cut-off age depicted as dashed vertical red line

Instead, parameters tuning is done manually using an iterative approach. The process involves running the LCS system and observe the final accuracy on testing data with only one parameter changed at a time. This allows comparison between the effect on accuracy for one parameters on different values.

After many trials and experiments, it is found out that out of all the 14 parameters needed to run LCS system, only a few of them have impact on the overall accuracy, namely Maximum Number Of Iteration, Maximum Population Size, Covering Wildcard Probability and Initial Range Factor. It is discovered that large number of population size alongside with a small value in initial range factor tend to produce good result, while a small number of population size and a big value in initial range factor can produce similar result. Also the mutation probability is found to have adverse effect on the accuracy when set to high(above 0.5), yet once it is below 0.1 it does not have an substantial influence on the performance.

Overall Maximum Number Of Iteration, Maximum Population Size and Initial Range Factor together has the biggest impact on the accuracy of the LCS system. The selection of those parameters also depends on the number of training sample. If the number of instances in the training sample is large, the number of iteration LCS should run needs to be increased as well.

6.2 Subsets of Classifications

First, the simplest case of classifying between only two classes of sound files was attempted. As shown in Figure 11, the overall accuracy of the LCS system converged on around 97% after 3,000 iterations. And as shown in the confusion matrix table, 97.4% of testing instance belonging to car and office was correctly classified. The LCS was not able to make a decision for approximately 2% of the testing instances.

Next the above experiment was extended by the addition the city centre data. The system converged on accuracy of 95% at around 1,000 iterations, representing a decrease by 2% relative to the previous testing result when only two classes were involved.

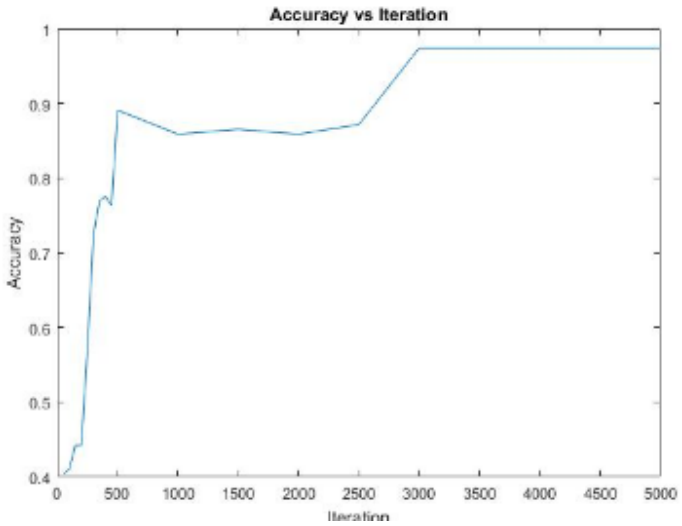


Figure 11: Accuracy with two classes over 5,000 iterations

Table 5: Two class confusion matrix

	Car	**Office**	**Uncovered**
Car	97.4	1.3	1.3
Office	0	97.4	2.6

It can be observed that the system has the best performance on classifying office, where 100% of the testing instances were correctly classified. The accuracy of classification on car has been reduced down to 88.5% from around 97% in the previous test.

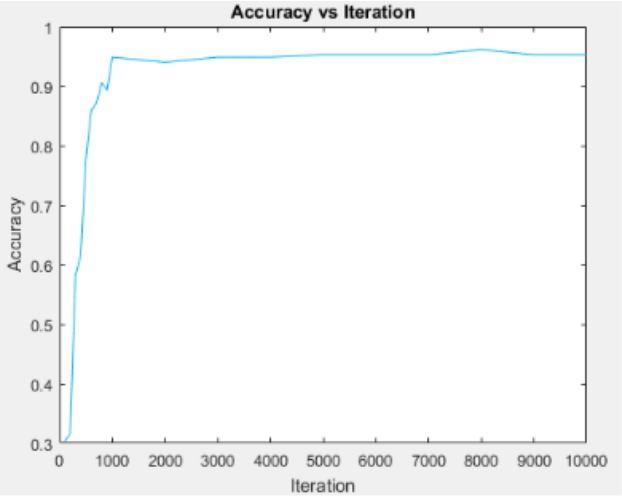


Figure 12: Accuracy with three classes over 10,000 iterations

Table 6: Three class confusion matrix

	Car	**Office**	**City Centre**	**Uncovered**
Car	88.5	5.1	5.1	1.3
Office	0	100	0	0
City Centre	2.7	2.7	93.4	1.3

The system converged on a much lower accuracy (80%) when data from four classes were used. A noteworthy observation is that the introduction of an additional class has little impact to the classes beforehand, yet as it can been seen in the confusion matrix, 34.2% of testing instance belonging to metro station was falsely classified as city centre.

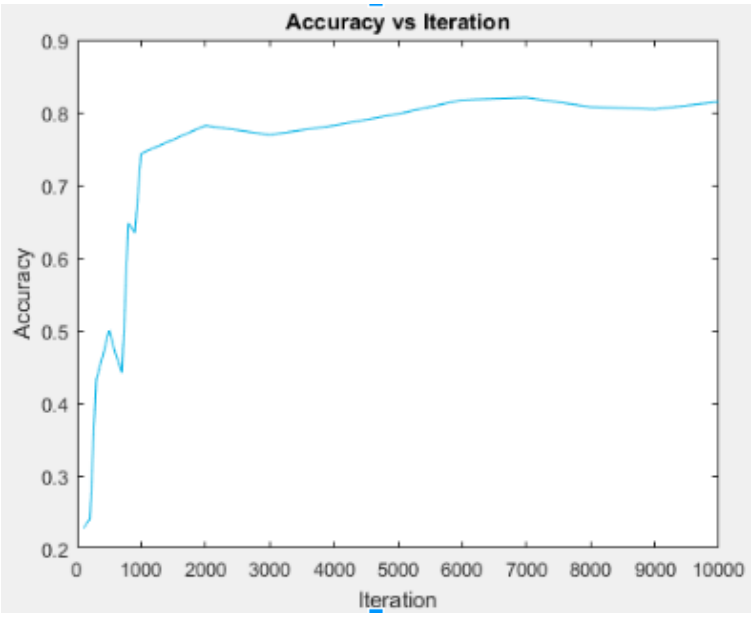


Figure 13: Accuracy with four classes over 10,000 iterations

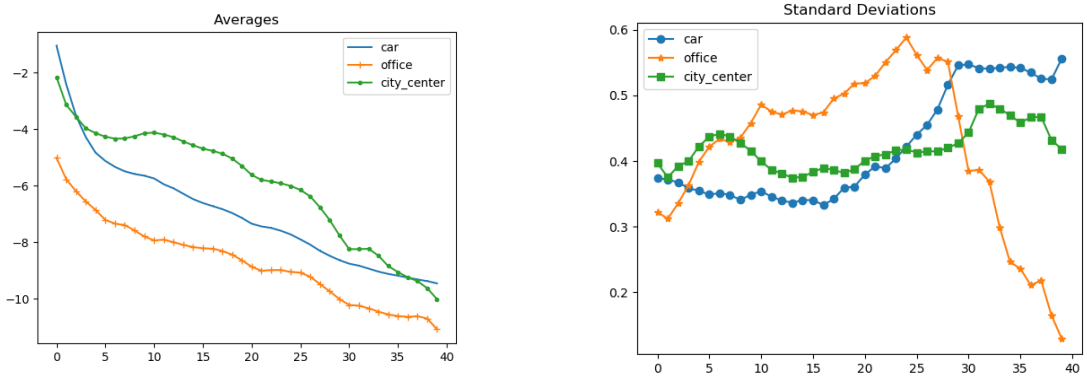
Table 7: Four class confusion matrix

	Car	**Office**	**City Centre**	**Metro Station**	**Uncovered**
Car	91.0	3.8	0	3.8	1.3
Office	0	96.1	0	1.3	0
City Centre	0	2.7	88.5	7.7	1.3
Metro Station	3.8	5.1	34.1	50.0	6.4

As shown in the previous three tests, as the number of classes in the testing data increases, the overall accuracy of the LCS system decreases. This is as expected as the introduction of more classes will inevitably introduce many noise to the classifiers as well. When all the classes in the testing dataset were used, it was observed that the overall accuracy decreased to around 44%.

7 Discussion

The LCS system demonstrated good classification accuracy when data of three classes are used. However, when class "metro station" is introduced, the accuracy was brought down to around 80%. This could be interpreted by looking at the characteristic curve for different classes.



(a) Means (b) Standard deviations

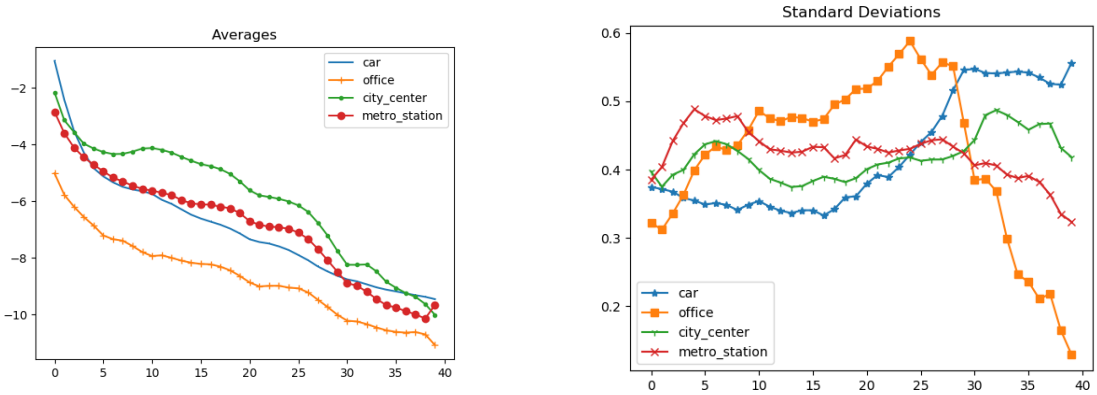
Figure 14: Average and standard deviation of sound file averaged mel bands across all sound files for three classes

Figure 14(a) is the average standard deviations for each mel band (attribute) across all instances in the dataset for class "car","office" and "city centre", while Figure 14(b) is the average means for each mel band (attribute) across all instances for the same classes. These two graphs represents the general trend of all instances belonging to those classes in the dataset. As it can be observed, those three lines in each of the figure are relatively distinct from each other, meaning that it is relatively easy for the LCS system to generate rules that can distinguish them from each other.

However, when class "metro station" are added in the dataset, as reflected in Figure 16 where the lines are not as distinct as before. The curve of city centre and metro station follow a similar trend, meaning that if the range covered by one classifier could potentially cover the curve for the other class as well. Therefore, it is unsurprising that metro station has a 35% chance of being falsely classified as city centre.

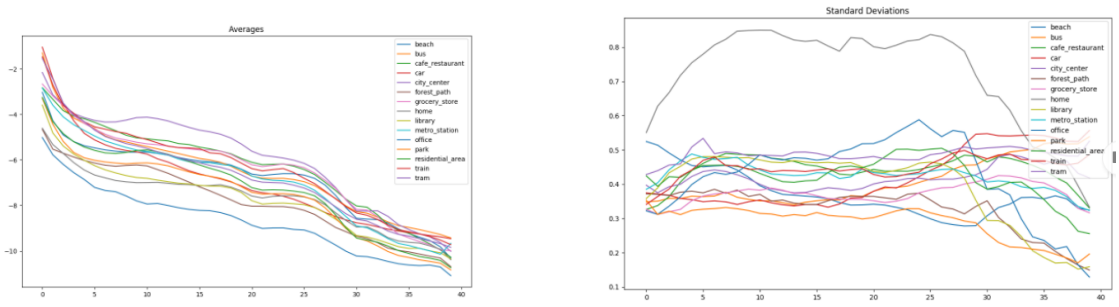
Figure 15 showcases the characteristics curve for all 16 classes. It is clear that the curve for many classes are clustered together, meaning that the classifiers in the LCS system have a very high chance of grouping the curve belonging to several classes into one set of rules. Thus the low accuracy of classifying 16 classes can be understood.

In general, the reduction of features means that a large proportion of information is thrown away. Given the fact that there are 80 attributes in a instance and each of them are continuous, the amount of training data (around 4,000) is insufficient for LCS to generate accurate rules to classify classes that has their characteristic curves clustered together as shown in the figures above.



(a) Means (b) Standard deviations

Figure 15: Average and standard deviation of sound file averaged mel bands across four classes of sound files



(a) Means (b) Standard deviations

Figure 16: Average and standard deviation of sound file averaged mel bands across all instances of all classes

8 Future Research

As previously discussed, quality of the results achieved by a machine learning system is highly dependent on the quality of the features used as input. While reducing the dimension of the data by taking the mean and standard deviation of the mel bands worked well for classification of a few different scenes, the classifier performed poorly when presented with more categories to discriminate. This behaviour indicates that the features used did not adequately capture the nuances separating different scenes and that representing the data in different ways may be beneficial to performance.

It may be useful to do Fourier analysis in order to better represent the shape of a sound file's magnitude-mel band curve, as the shape appears to be a distinctive feature for many of the instances. While this could be useful, more research would be needed to determine a means of representing a Fourier series in a way that doesn't increase dimensionality too drastically.

The DCASE Baseline system provides numerous different ways of extracting features from sounds files, so another avenue of future research would be to test the performance of different types of features extractable with the system. Potential experiments could be conducted into making use of mel scale cepstral coefficients rather than pure mel scale band data as well as experimenting with different window lengths, hop lengths and mel band sizes. A neural network or genetic algorithm could be used to optimise the feature extraction as well as LCS parameters in order to find the optimal settings across both systems.

Smoothness of the different magnitude-mel band curves would be an interesting feature to look into using for future experiments. Features could potentially be derived from the location of maxima and minima as well as the smoothness between adjacent mel bands.

There might also be potential for research into swapping the data and plotting the magnitude over time of each mel band. Upon inspection, patterns do not immediately emerge as with magnitude over mel band of each time slice, but further research may find significant features to be extracted from that data. Example plots of magnitude over time for different mel bands can been seen in Figure 17.

Given time for further research, investigation into classification using the high dimensional data directly extracted using the DCASE Baseline system's feature extractor also has potential to yield better results. Retaining all of the time data would make classifiers in the LCS more prone to overfitting an instance, but increasing generality with a higher proportion of wildcards may be able to offset this while also allowing useful information to be exploited rather than lost. This would also require a larger training dataset in order to be properly tested, as more data points would be needed to describe the higher dimensional space.

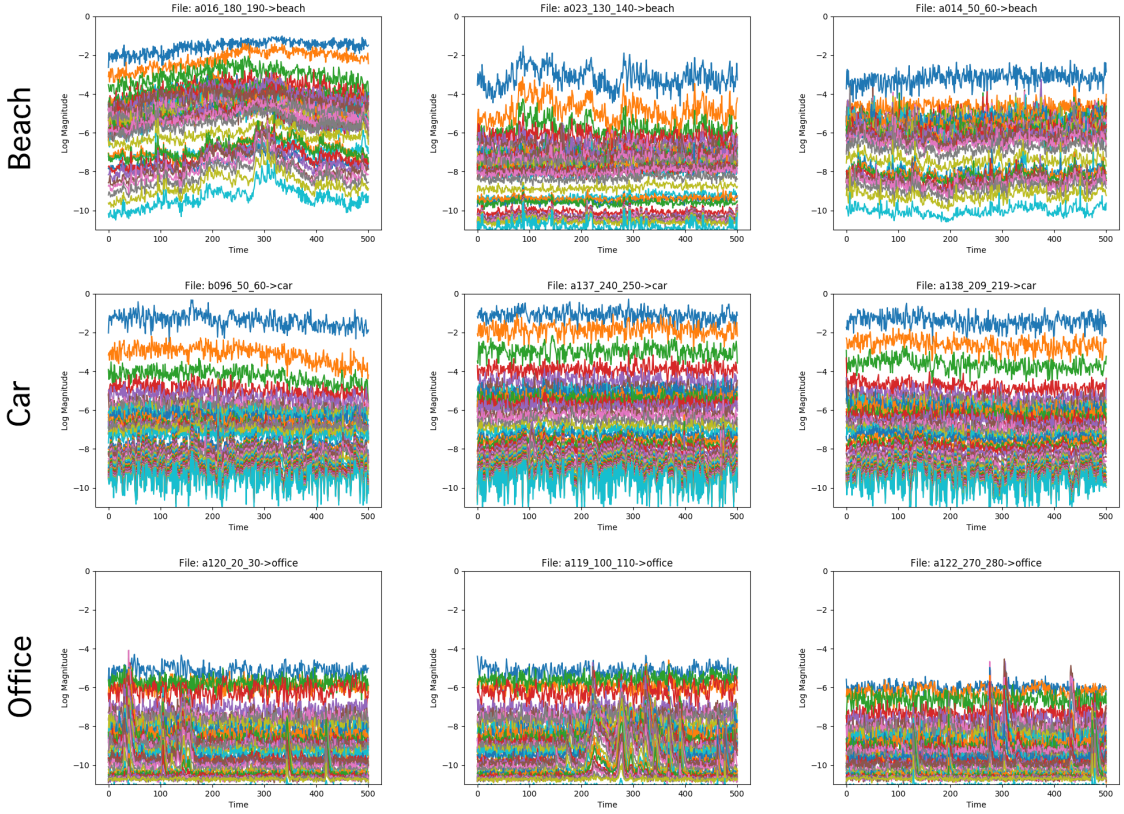


Figure 17: Each mel band shown over time for three sound files from three classes

9 Conclusion

Acoustic sound classification is ongoing field of research in which machine learning techniques have recently been making fast progress. The annual Detection and Classification of Acoustic Scenes and Events (DCASE) challenge provides various sound related tasks for competitors to train models on in attempts to push the boundaries of machine sound recognition performance. The challenge provides a training data set of sound files as well as a Baseline system that performs feature extraction by decomposing audio data into usable features. Taking a different direction to the neural networks that are most commonly used in this competition, learning classifier systems (LCSs) were used to explore their suitability to the given task of acoustic scene classification. Using the feature extractor provided, a feature set comprised of the average magnitude for 40 mel bands over the sound files was used to train the LCSs. It was found that while the system would perform extremely well when discriminating between a small set a classes, performance declined steeply as the number of classes increased and the patterns characterising them become less distinctive. It is potentially due to the method of dimension reduction used that the LCSs struggled to discriminate between larger sets of classes, so further research is required to investigate alternative feature sets and matching LCS parameters settings that may allow the systems to achieve greater performance in this area.

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] Ryanurbs. (2016) Ryanurbs/ELCS. [Online]. Available: https://www.github.com/ryanurbs/eLCS

[19] ScottMcCormack. (2017) ScottMcCormack/CITS4404. [Online]. Available: https://www.github.com/ScottMcCormack/CITS4404

[20] 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

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

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

[23] 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.