CI Rainbow:
Modeling Wildlife Population with Sound Identification

A Thesis Presented to

The Faculty of the Computer Science Program

California State University, Channel Islands

In (Partial) Fulfillment

of the Requirements for the Degree

Masters of Science in Computer Science

by

Janeth Moran-Cervantes

May 2018

CI Rainbow:

Modeling Wildlife Population with Sound Identification

A Thesis Presented to

The Faculty of the Computer Science Program

California State University, Channel Islands

In (Partial) Fulfillment

of the Requirements for the Degree

Masters of Science in Computer Science

by

Janeth Moran-Cervantes

May 2018

APPROVED FOR THE COMPUTER SCIENCE PROGRAM

5/23/2018

Professor: Dr. Andrzej Bieszczad Date

5/23/2018

Professor: Dr. Jason Isaacs Date

5,23,2018

Professor: Dr. Houman Dallali Date

APPROVED FOR THE UNIVERSITY

May 24, 2018

Dr. Joe Shapiro Date
Interim Dean and AVP of Extended University and International Programs

CI Rainbow:

Modeling Wildlife Population with Sound Identification

by

Janeth Moran-Cervantes

Computer Science Program

California State University Channel Islands

Abstract

The CI Rainbow system is a cloud-based data collecting infrastructure that consists of sensor nodes, distributed throughout a terrain, that collects data such as temperature, humidity, moisture, image, video, sound, and RF tag readings. The system is a self-sustainable framework with flexible sensor node additions, removals, and configuration modifications currently developed at the University Park with the ultimate deployment goal at the Santa Rosa Island. In this paper, we provide a framework for creating sound identification models and using those models to identify wildlife. We use a dataset with 625 training samples from 5 classes, extract Mel Frequency Cepstral Coefficient features using 13 coefficients, apply Principal Component Analysis for dimension reduction, evaluate the models using 5 metrics (*accuracy*, *precision_ macro*, *recall_ macro*, *f1_ macro*, and *f1_ weighted*), and perform hyperparameter optimization with grid search and cross validation to create a Support Vector Machine classification model. We trained on 80% of the dataset and obtained an optimal model with an rbf kernel, 9 principal components, $\gamma = 0.01$, and $C = 10$ that classified with 0% error on the remaining 20% of the dataset. Lastly, we outline how this framework will be integrated with the existing CI Rainbow infrastructure to create visualizations of population models that will help in the effort of preserving wildlife on the Santa Rosa Island.

Acknowledgements

We gratefully thank Dr. AJ Bieszczad for his support and guidance on this project.

CI Rainbow:
Modeling Wildlife Population with Sound Identification

by

Janeth Moran-Cervantes

Computer Science Program

California State University Channel Islands

Abstract

The CI Rainbow system is a cloud-based data collecting infrastructure that consists of sensor nodes, distributed throughout a terrain, that collects data such as temperature, humidity, moisture, image, video, sound, and RF tag readings. The system is a self-sustainable framework with flexible sensor node additions, removals, and configuration modifications currently developed at the University Park with the ultimate deployment goal at the Santa Rosa Island. In this paper, we provide a framework for creating sound identification models and using those models to identify wildlife. We use a dataset with 625 training samples from 5 classes, extract Mel Frequency Cepstral Coefficient features using 13 coefficients, apply Principal Component Analysis for dimension reduction, evaluate the models using 5 metrics (*accuracy*, *precision_macro*, *recall_macro*, *f1_macro*, and *f1_weighted*), and perform hyperparameter optimization with grid search and cross validation to create a Support Vector Machine classification model. We trained on 80% of the dataset and obtained an optimal model with an rbf kernel, 9 principal components, $\gamma = 0.01$, and $C = 10$ that classified with 0% error on the remaining 20% of the dataset. Lastly, we outline how this framework will be integrated with the existing CI Rainbow infrastructure to create visualizations of population models that will help in the effort of preserving wildlife on the Santa Rosa Island.

Acknowledgements

We gratefully thank Dr. AJ Bieszczad, thesis advisor, for guidance and inspiration on this project. Completion of this thesis would not have been possible without his consistent support. We also thank Dr. Jason Isaacs and Dr. Houman Dallali for being part of the thesis committee and providing valuable feedback.

Contents

List of Figures

Listings

1 Introduction

1.1 Introduction to CI Rainbow Project

The Santa Rosa Island (SRI) is one of the five islands of the Channel Island National Park. Isolation has caused the island to house unique plant and animal species. In addition, the island contains "irreplaceable archeological resources, and important geological and paleontological specimens" [3]. Hence, it is of high importance to preserve the habitat present in the island. The CI Santa Rosa Island Research Station (SRIRS), located in the SRI, is a newly established research station (established in 2012) aimed at providing and supporting research, education, and outreach activities [4]. Monitoring wildlife is part of the effort of preserving flora and fauna. An important aspect of wildlife monitoring is building population models, to determine where animals are at any given moment, and migration models, to understand animal movement and behavior patterns. Traditionally, human-run surveys are involved but that has a lot of shortcomings including cost, coverage capabilities, errors, and time effort, leaving researchers less time for analysis.

There are means to conduct automated visual surveys, but they have flaws as well, namely, a limited field of view. Yet another approach may be RF-tracking, but there are problems with tagging and with the range of RF communication. With a highly accurate sound identification model in place, audio-based surveys address some of the shortcomings of the previous approaches. Audio recorders are (1) resilient to light conditions, (2) omni-directional, (3) concealed, (4) non-intrusive, and though sound also has a range limitation, the coverage area is wider than compare to the previous approaches described. Such a system requires a physical and software infrastructure distributed throughout the terrain.

The CI Rainbow system is a cloud-based data collecting infrastructure that consists of sensor nodes, distributed throughout a terrain, that collect data such as temperature, humidity, moisture, image, video, sound, and RF tag readings. The sensors are powered by solar panels and communicate through a number of terrestrial and aerial links to a main data center. The system is a self-sustainable framework with flexible sensor node additions, removals, and configuration modifications. System management and data collection is accessible through a web-based application. The ultimate goal is to have the CI Rainbow system deployed on the SRI [5]. This will be accessible to researchers and educators interested in conducting any studies of the Channel Islands. The data collected will allow for machine learning and data mining techniques to be employed for different studies and applications. In particular, the sound recordings are collected, identified, and used to model wildlife population.

Tiered deployment consists of (1) simulation in a lab setting for proof of concept, (2) deployment at the University Park (UP) at CSUCI, and (3) infrastructure and system deployment at SRI. The current state

of the CI Rainbow system is in development in a lab environment. Deployment at the University Park at CSUCI will be a testbed for continued test and development as it has similar settings and conditions as those in the SRI.

1.2 Remaining Chapters

This paper focuses on building a framework for sound identification for the purpose of modeling wildlife population. First, we provide a brief background on audio structure and machine learning techniques for audio processing and sound identification. We then provide a summary of previous classification studies conducted. Then, we discuss the methodology used for sound identification for modeling wildlife population. We describe the current state of research and development. And lastly, we provide our future work and continued studies.

1.3 Key Terms

classification: the process of classifying objects into categories

machine learning: the study of learning from data

mp3: MPEG-2 Audio Layer III; audio coding format for digital audio that uses lossy data compression

PCA: Principal Component Analysis, a dimension reduction technique

supervised learning: "the machine learning task of inferring a function from labeled training data" consisting of pairs of input objects and a desired output value [6]

SVM: support vector machine, a supervised machine learning algorithm used primarily for classification

unsupervised learning: "the machine learning task of inferring a function to describe hidden structure from unlabeled data" [7]; the training data consists of only input objects

WAVE/WAV: Waveform Audio File Format; a uncompressed, lossless audio file format standard for storing an audio bitstream on PCs developed by Microsoft and IBM [8]

YAAFE: Yet Another Audio Feature Extractor; library used to extract features from audio files

2 Field Overview

Machine learning is the study of learning from data. While there are various paradigms that focus on learning from data, such as statistics, supervised learning, unsupervised learning, reinforcement learning, and data mining to name a few, we focus on the supervised learning paradigm for the application in wildlife sound identification. Concretely, we follow the classification workflow outlined below for applying supervised learning techniques in order to identify wildlife from audio recordings.

2.1 Classification Workflow

Identifying an object is equivalent to classifying or categorizing that object. This process consists of an offline process that trains and generates a machine learning model and an online process that uses the trained model to identify the unknown object. Figure 1 shows the workflow for the model creation and identification processes.

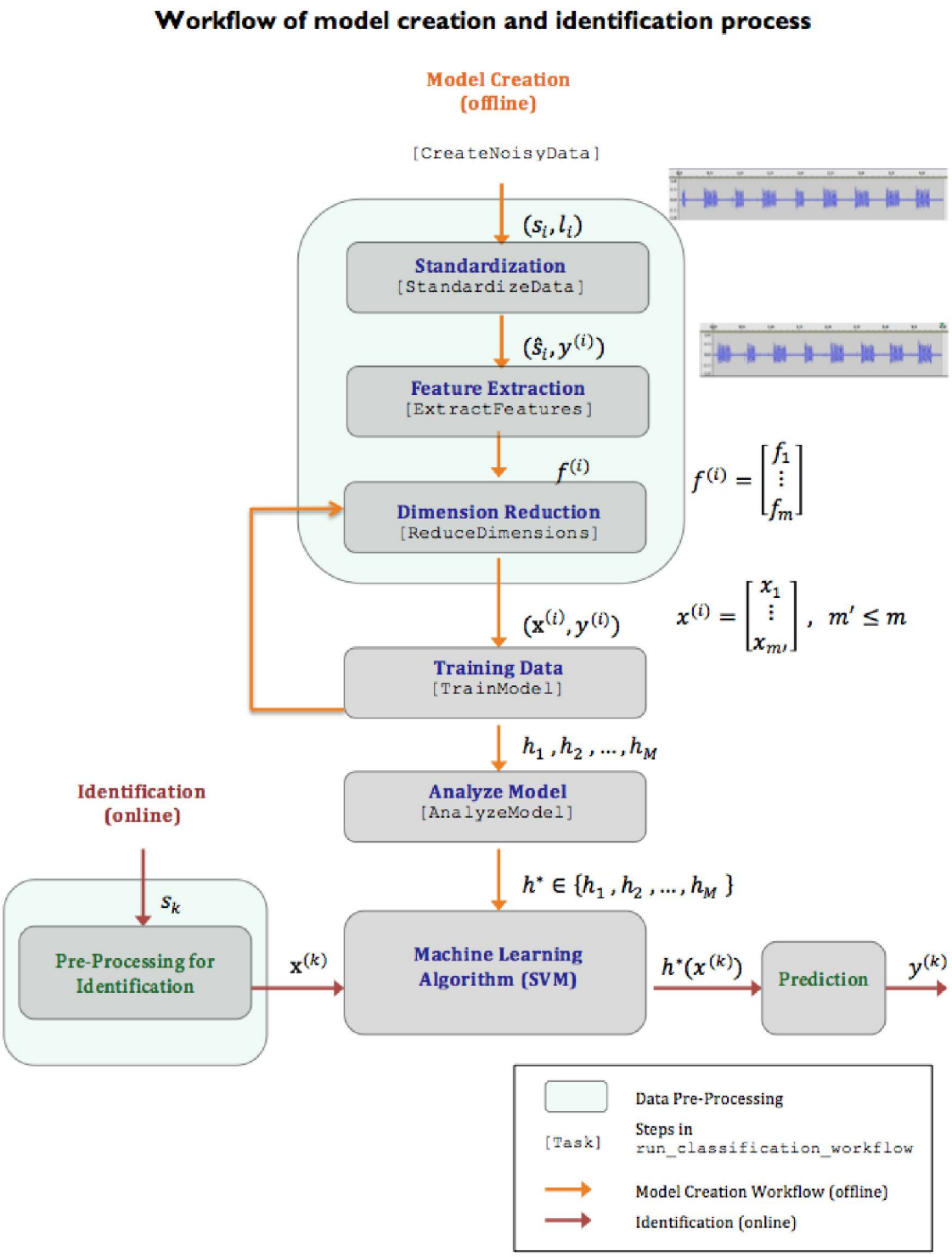


Figure 1: Workflow Process: (Offline) Model Creation, (Online) Identification

2.1.1 Data

We gather samples of objects to be classified and their corresponding classification label. Each sample s_i has the category label l_i. For example, s_i can be a book object, and l_i the corresponding genre label. In

the context of wildlife identification, s_i is an animal sound (WAV) recording and l_i is the corresponding animal type (e.g. seagull).

2.1.2 Data Cleaning and Standardization

The data $\{(s_i, l_i)\}$ is then processed for cleaning and standardization. In this step, data that is corrupted is removed or cleaned. Missing values may be filled with default values. Audio files are cropped to a standard length, resampled at a standard sampling rate and converted to a standard number of channels. Noise reduction or removal techniques can also be applied in this step. The labels are transformed to a numerical value representation. The result is the set of standard samples $\{(\hat{s}_i, y^{(i)})\}$. We standardize the audio files to channels=1, samplerate=44100Hz, and length=30secs.

2.1.3 Feature Extraction

In the feature extraction step, each sample is applied one or more techniques that generate features. These techniques are varying forms of data transformations, $T_j(\hat{s}_i) = f_j$ where $\hat{s}_i \in \mathbb{R}^n$ with n being the total number of samples in the recording, and $f_j \in \mathbb{R}^{m_j}$ with m_j being the number of features extracted when applying T_j. These transformations may be dependent on the sample being processed. For example, color histograms and wavelets may be more applicable for images while amplitude modulation and Mel-frequencies cepstrum coefficients may be more applicable to sounds. The generated features, $\{f_k\}$, combined form a feature vector, $f^{(i)} \in \mathbb{R}^m$, corresponding to the sample $\hat{s}_i$. That is,

$$T(\hat{s}_i) : \mathbb{R}^n \to \mathbb{R}^m \tag{1}$$

2.1.4 Dimension Reduction

The high dimensional feature vectors are then reduced for computational efficiency during model training and later, classification. Principal Component Analysis (PCA) is a dimension reduction transformation, R, that maps the feature vector to a lower dimensional space. Let $f^{(i)} \in \mathbb{R}^m$ be a feature vector. Then, $R(f^{(i)}) : \mathbb{R}^m \to \mathbb{R}^{m'}$ where $m' \leq m$. The value of m' is determined by the weights of the eigen values of the covariance matrix of the training data.

2.1.5 Model Training

While different types of models such as Dynamic time warping, Hidden Markov models, Artificial Neural Networks, and Support Vector Machines, can be used, careful inspection and study is necessary in order to achieve the best results on the given dataset as some models may perform optimally in a particular domain and poorly in others. The classifier is then trained with q training samples, $(x^{(i)}, y^{(i)})$ where $i \in \{1, 2, ..., q\}$. The output is an optimal function, h^*, that maps a feature vector to one of the known target values.

2.1.6 Classification

To classify an unknown object, s_k, we apply the same transformations that were applied to the training data. That is, s_k is standardized, features are extracted and the corresponding feature vector is reduced to the same dimension as the training data. See Identification workflow in Figure 1. The transformed vector, $\mathbf{x}^{(i)}$, is then passed in to the model, $h^*(\mathbf{x}^{(i)})$, and the model outputs a prediction, $y^{(k)}$. That is, the optimum trained model, h^*, identifies the unknown object s_k as $y^{(k)}$.

2.1.7 Evaluation

The accuracy of the hypothesis, h^*, is dependent on the pre-processing of the data, the type and amount of features used, the appropriateness of the model selected, how well tuned the model parameters are and most importantly, the quality and amount of data provided. One way to evaluate the model is to use k-fold cross validation coupled with the percent error as a measure. Using the percent error as a metric has the property of being biased towards classes with a larger number of training data. In such cases, the quantity and data diversity may favor a particular class or set of classes. A model that is not properly trained, or that consistently "classifies" objects to target values that are most likely to appear can have a high classification accuracy (small classification error), yet not have learned well or anything at all. In our evaluation, the model is trained on a given training dataset using k-fold validation.

Precision and recall metrics are calculated on a test dataset to evaluate the model. Precision is defined as number of true positive divided by the number of predicted positives (true positives + false positives). Recall is defined as the number of true positives divided by the number of actual positives (true positive + false negative), see Fig 2 [1]. These metrics address accuracy and coverage and are consolidated into a single value by the F1 score metric [9].

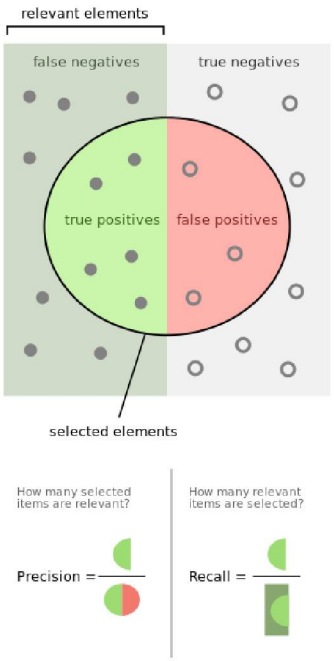


Figure 2: Precision and recall [1]

2.2 Previous Studies

In 2013, we conducted a study in which we tested different types of classifiers on a given dataset and measured their performance in order to determine which classifier generated the model with the lowest classification error [2]. Given nine 17-dimensional data samples, each with a unique target value, we generated training data by applying different levels of noise to the original nine samples. We classified into one of the nine known target classes using various dimension reduction and classification techniques. The classification techniques used were: (1) linear regression, (2) logistic regression, (3) k-means, and (4) SVM. Each technique generated a model from variant implementations and data transformations, namely, LDA and PCA for data dimension reduction.

Noisy data was generated by applying noise generated from a normal distribution with variance, σ^2, to the original samples. This was then used to train the classifier and create a model. We trained with 5-fold cross-validation using 1000 noisy data samples per target class. Our experiments confirmed SVM to be the most accurate out of the four distinct machine learning techniques (see Figure 3) with 0% classification error (100% correct classification).

Technique	Variations	Results (sigma, Error)
Linear Regression	One Step Learning	0-0.3; **0-240** (sq err, exponential), ~**0.55-0.85**
	Gradient Descent	0-0.3; ~**5e^{279}** (sq err, overflow)
	PCA	0-0.3; **0.55-0.85**
Logistic Regression	Quasi-Newton Method	0-0.3; **0-0.7** (logarithmic growth)
	PCA, Quasi-Newton Method	0-0.3; 0-0.3 (3 feats), **0-0.65** (5 feats)
	PCA, Quasi-Newton Method, learning rate (α) cross-validation	α in {0.001, 0.001, 0.01,0.1, 0.5}, e=**0.005** α=0.4-1.9, e<**0.005**
	LDA, Quasi-Newton Method, learning rate cross-validation	α in {0.001, 0.001, 0.01,0.1, 0.5}, e=**0.005** α=0.4-1.9; e<**0.005**
k-Means	PCA	• Clustering dependent on initial centroids => initialize centroids using training data • 2 dimensions, not enough • 3 dimensions, classes well separated
	LDA	
SVM	PCA, RBF Kernel, (C, γ) cross-validation	γ<0.4, C< 0.28, e<**0.9** (*large error*) $\gamma\geq$0.4, C$\geq$ 0.28, e<**0.001**
	PCA, RBF, full training data	$\gamma\geq$0.4, C$\geq$ 0.28, e=**0**

Figure 3: Performance results of 4 machine learning techniques from the study in [2]

The optimal model was an SVM with a Radial Basis Function (RBF) kernel, $\gamma \geq 0.4$, and $C \geq 0.28$ parameters, and the full training data resulted in 0% classification error.

2.3 Sound Identification: Environment, Music, Speech

There have been multiple studies on sound recognition in the music and speech domain given their practical application in every day life. Environment sound is often filtered out as noise in order to focus on the object of interest. Cowling and Sitte [10], in a comprehensive comparison study, compared different techniques for speech and musical instrument recognition with the purpose of determining suitability in environment sound recognition. Their results indicate that "techniques traditionally known as best performers in the speech/speaker or in the musical instrument recognition scenario, are either not suitable, or not as good performers for the neglected field of environment sound recognition" [10]. They compared classification and feature extraction techniques for environmental sound recognition and their results showed Dynamic Time Warping to be a high performing classifier while Gaussian Mixture Models (GMM) to be average in performance. Stationary (e.g. Mel frequency cepstral coefficients) and non-stationary (e.g. Short-time Fourier Transform, Fast Wavelet Transform) features were applied. Dynamic time warping paired with either Mel frequency cepstral coefficients or continuous wavelet transform resulted in the highest classification accuracy.

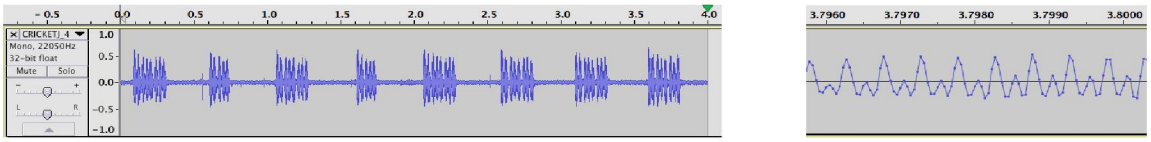
Other studies in the bioacoustics domain focus on identifying specific wildlife such as in [11] where Acevedo used support vector machines, decision trees, and linear discriminant analysis to classify bird and amphibian calls. Their automated classification technique was incorporated into an automated

digital recording system. Similarly, this paper focuses on incorporating classification into the CI Rainbow system. In particular, in this paper we present a framework used to generate classification models that can be incorporated into a general system. The framework is an extensible, configurable, standalone classification workflow that can be applied to different domains and integrated with different applications.

In [10], GMM was used over SVM due to GMM's popularity in previous applications. Based on this study, our previous studies, and its increase popularity, SVM is hence, a comparable high performing classifier that we focus on in this paper as a baseline for the framework that we present.

2.4 Audio Sound

An audio recording is a discrete approximation of a sound wave (see Figure 4). The recording is composed of approximated sampled data points of a sound wave over a period of time, where the height of the data point (or amplitude of the wave), is the measurement of the sound pressure level measured in decibels, dB. The more samples gathered, the better the approximation. The sampling frequency, measured in Hz, is the number of samples taken per second. The bit depth determines the precision of the measured sample data point, typically 8-bit or 16-bit.



(a) 4 second sound recording

(b) zoomed in sound recording

Figure 4: Sound waves of sound recordings

Mel Frequency Cepstral Coefficients (MFCC) is a feature extraction technique that has been used both in music and speech recognition and in Cowling's studies, MFCC and Continuous Wavelet Transform (CWT) were found to be the most informative feature extraction techniques. In [12], support vector machines were used in conjunction with MFCC to identify bird songs. In this paper, we focus on a framework for identifying different wildlife. While we focus on MFCC to extract features from the audio dataset, the framework that we present provides the functionality to experiment with additional features. The framework provides a baseline for automating the research component of model creation with the functionality to experiment with other features from the YAAFE library, as well as the ability to develop and incorporate custom ones.

3 Technical Details of the Work

3.1 Overview

We developed the Classification Workflow library (`mwlpsi-lib`), using Python 2.7.13, that is structured to facilitate future development. We include sample data, documentation, and the library modules.

3.2 Data: Audio files

We compiled standard uncompressed .wav audio recordings of bird species that could potentially be found at the SRI, namely, (1) bald eagle, (2) black caracara, (3) common wood pigeon, (4) fox sparrow, and (5) great tinamou [13]. These recordings were used for development, test, and analysis.

3.3 Libraries

The module that we developed uses Python 2.7.13. The Standardization step uses the Sound eXchange (SoX) 14.4.2 library, a command line utility implemented in C, to trim and resample the audio files at a specified length and rate respectively [14]. The Python `scikits-audiolab` 0.11.0 library was used to add noise to existing audio samples [15]. YAAFE 0.64, an audio features extraction Python toolbox, was used for extracting features from the audio files with dependencies on C libraries (`libsndfile` 1.0.25.0, `libmpg123` 1.24.0.0, `HDF5` 1.10.0, `lapack` 3.6.1, and `FFTW3` 3.3.5) [16]. The Python `scikit-learn` 0.19.0 library was used to apply dimension reduction techniques (i.e. decomposition), model training (svm), and analysis (pipeline, model_selection, and metrics) [17]. The Python `mwlpsi-lib` module that we developed was tested on macOS 10.12: Sierra on a 2.8GHz Intel Core i7 processor with 16 GB 1333 MHz DDR3 Memory.

3.4 Configurations

Given the varying lengths of recurrent patterns in wildlife natural sounds, the limited recordings available for training data, and the varying sources from which the recordings were obtained, all recordings were standardized to 30 second time-length, 16-bit bit-depth, 1 channel recording, and 44,100Hz sampling rate for this analysis. However, the recording configuration of the sound recording sensor nodes in the CI Rainbow infrastructure can be updated. The 30 second time length implies higher likelihood of multiple recurrent patterns occurring in a given sample. A 16-bit (vs. 8-bit) bit depth is used to capture precision of the sound wave amplitude measured in decibels. Recordings are standardized to 1 channel to simplify

and maintain consistency across variant recordings. While humans hear sounds best from 1,000 Hz to 5,000 Hz, where human speech is centered, humans can in fact hear sounds at frequencies from about 20 Hz to 20,000 Hz [18]. It is standard to assume that frequencies above 22,050Hz cannot be heard by humans. By Nyquist-Shannon sampling theorem, we should sample at twice the maximum frequency (22,050 * 2 = 44,100) [18]. For this reason, most of the recordings acquired from external sources were sampled at 44,100Hz while a few were sampled at a higher rate (i.e. 48kHz) [13]. To achieve parity and comparable results, we standardized the recordings to a sampling rate of 44,100Hz, though in practice, this range can be modified to encapsulate frequencies outside of this range as many animals (i.e. porpoise has a hearing range of 75-105kHz) use different frequency ranges to communicate [19].

3.5 Processing Audio

In section 4.2, we provide an overview of how individual steps in the workflow process outlined in section 2.1 are executed. Here, we provide details of extracting features and building the feature vectors.

The YAAFEE library provides 26 available feature extraction techniques. Some techniques are derivatives of others while particular ones were developed for specialized applications. One of the techniques included is Mel Frequency Cepstral Coefficients (MFCC), which generates features through a series of steps that include the following. (1) Break the signal into frames. (2) Compute a power spectrum estimate using a Discrete Fourier Transform. (3) Apply the mel filterbanks, where each filter is a triangular filter with height $2/(frequency_{max} - frequency_{min})$, to the power spectrum by multiplying each filter bank with the power spectrum (YAAFE uses 40 filters). Then, sum the energies to get an estimate of how much energy was in each filterbank. (4) Apply the log of each of the 40 energies. (5) Take the Discrete Cosine Transform (DCT) of the 40 log filtered energies. (6) Keep 2-13 DCT coefficients, discarding the rest [20]. Keeping 13 coefficients in the last step will generate 13 features per frame.

Given a standardized recording as described in section 3.4, $\hat{s}_i$, when MFCC with block size of 512, step size of 256, and 13 Cepstral coefficients is used to extract features, the standard audio (16-bit, 30secs, 44100Hz) is divided into 5186 frames each with 13 features. This yields a total of 67184 numerical values. Hence, the feature vector for $\hat{s}_i$ with only MFCC features extracted is $f^{(i)} = [f_1^{(i)} f_2^{(i)} ... \ f_{67184}^{(i)}]^T$ (see the left images on Fig 6 (a)-(e), where the first 200 MFCC features are shown for each of the 5 samples within each of the 5 classes). The first set of 13 features are features from the first frame, the second set of 13 features are from the second frame, and so on. Adding additional features will result in a higher dimensional feature vector. This is a high dimensional space and computationally expensive. Thus, we apply Principal Component Analysis (PCA) to reduce the dimensions of the dataset. Note that if no PCA were applied, we would have $f^{(i)} = \mathbf{x}^{(i)}$ in Figure 1.

In PCA, the eigen values and eigen vectors of the covariance matrix of the training data determine how distinguishing the samples are after projecting them onto the dimensions defined by the corresponding eigen vectors. Determining how many features to use can be done programmatically by projecting the test samples onto smaller dimensions using the dominant eigen vectors, calculating the classification error, and selecting the minimum number of dimensions that resulted in the largest precision and recall values. In this experiment, we manually selected different dimensions for reducing the training data and analyzed the results. We then performed hyperparameter optimization where one of the parameters that we optimized for is the number of principal components used to reduce dimension of our dataset.

4 Experiments

4.1 Setup

We explored different development environments and found PyCharm Community Edition 2016.3.2 to be the most useful, allowing to step through the code for debugging. Additional development and experimentation were done through terminal and Jupyter Notebooks.

4.2 Workflow

The Model Creation process outlined in Figure 1 was broken down into independent tasks. Doing so allowed for faster test and development as it helped us focus on individual tasks, abstract from other components, and narrow down issues. After development, modular components helped us perform faster analysis and experimentation. Creating a classification model is computationally expensive and it is inconvenient to spend resources (e.g. time) running through the entire workflow and obtain undesirable results due to incorrect use of parameters. In particular, creating classification models entails processing a large amount of data and analyzing different hyperparameters for improvements. Our implementation gives us the flexibility to validate the steps independently, catch issues, and experiment faster. A script based implementation makes the library easy to integrate with other programs or environments.

In the following subsections, we describe how each of the tasks, specified in brackets (i.e. `[CreateNoisyData]`) in Figure 1 as part of the model creation process, is executed. We describe the command line and .json configuration parameters required in each of the tasks. For example, one of the parameters required by the `CreateNoisyData` step is a struct with the key `seed_training_data` within the .json configuration.

4.2.1 Standardization

The standardization step is executed using the following command:

```
> python ./mwpsi/run_classification_workflow.py \
        -c ./tst/jmcModelConfig.json \
        -tasks CreateNoisyData,StandardizeData
```

Listing 1: Execute CreateNoisyData and StandardizeData

In Listing 1, the script `run_classification_workflow.py` is executed with parameter `-c` used to indicate the configuration file to be used (`jmcModelConfiguration.json`) as input and parameter `-tasks` to indicate the tasks to execute.

```
{
  "tasks" : ["CreateNoisyData,StandardizeData"],
  "seed_training_data": {
    "type": "audio",
    "classes": {
      "c1" : [
        "/<path_to_dir>/mwpsi-lib/data/trainingData/c1/bald_eagle_1.wav",
        "/<path_to_dir>/ms-mwpsi/mwpsi-lib/data/trainingData/c1/bald_eagle_5.wav"
      ],
      "c2" : [
        "/<path_to_dir>/ms-mwpsi/mwpsi-lib/data/trainingData/c2/black_caracara_1.wav",
        "/<path_to_dir>/ms-mwpsi/mwpsi-lib/data/trainingData/c2/black_caracara_2.wav"
      ],
      "c3" : [
        "/<path_to_dir>/ms-mwpsi/mwpsi-lib/data/trainingData/c3/common_wood_pigeon_1.wav",
        "/<path_to_dir>/ms-mwpsi/mwpsi-lib/data/trainingData/c3/common_wood_pigeon_2.wav"
      ],
      "c4" : [
        "/<path_to_dir>/ms-mwpsi/mwpsi-lib/data/trainingData/c4/fox_sparrow_1.wav",
        "/<path_to_dir>/ms-mwpsi/mwpsi-lib/data/trainingData/c4/fox_sparrow_2.wav"
      ],
      "c5" : [
        "/<path_to_dir>/ms-mwpsi/mwpsi-lib/data/trainingData/c5/great_tinamou_2.wav",
        "/<path_to_dir>/ms-mwpsi/mwpsi-lib/data/trainingData/c5/great_tinamou_4.wav"
      ]
    },
    "standardize_audio_parameters": {
      "mode" : 0,
      "channels" : 1,
      "samplerate" : 44100,
      "time_in_seconds" : 30
    },
    "random_data_parameters": {
      "mu": 0,
      "sigma" : 0.01,
      "quantity": 3
    },
    "workspace" : "/<path_to_dir>/workspace/"
  },
  "standardize_data": {
  },
}
```

Listing 2: jmcModelConfig.json - Input for Standardization step

Listing 2 shows the contents of the configuration file containing the parameters required to process the `CreateNoisyData` task. In the configuration, the `standardize_audio_parameters` specifies to standardize each sample to 1 channel, 44100Hz sampling rate, and 30 seconds in length. The `random_data_parameters`

specifies to generate 3 noisy training samples from each original sample (.wav file) in the `seed_training_data`. Noisy samples are generated by duplicating the seed sample and adding noise randomly generated from a normal distribution with $\mu = 0$ and $\sigma = 0.01$. These values were selected as a baseline for introducing data variation into the model creation process as the noise in practice can vary. In such cases, we need to study the type of noise present in the environment and consider applying noise reduction techniques. The standardized and noisy files aren then written to the path defined in the `workspace` parameter and are under the folder names labeled in the following form:

`training_data.<YYYY-MM-DD_HH.MM.SS>.c<channels>.r<samplingRate>.s<lengthInSeconds>.std`

and

`training_data.<YYYY-MM-DD_HH.MM.SS>.c<channels>.r<samplingRate>.s<lengthInSeconds>.noisy.mu.<muValue>_sigma.<sigma/value>.c<numberOfClasses>.n<noisySamplesPerTrainingSample>`

Figure 5 shows the directory structure of the standardized and noisy samples that are generated by the tasks specified in Listing 1.

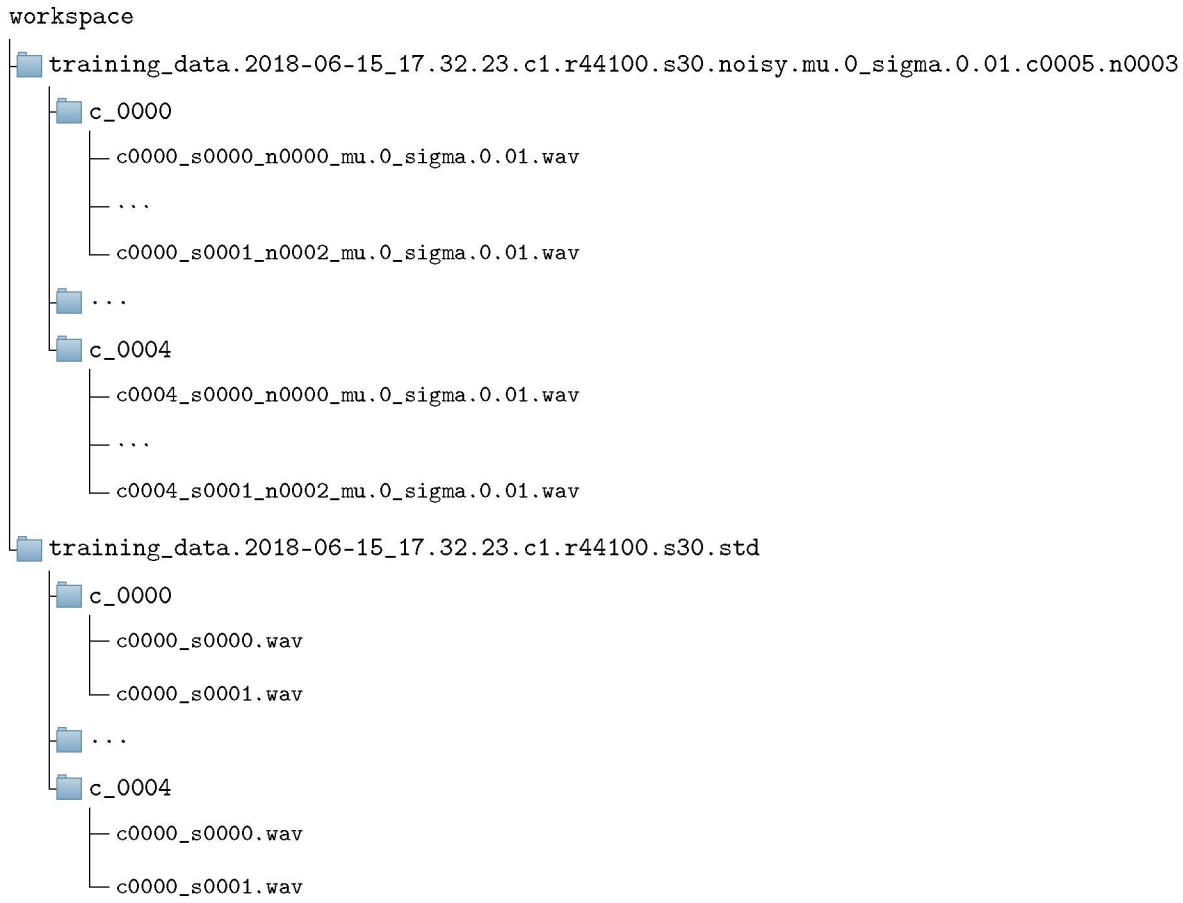


Figure 5: Noisy and Standardized Samples

4.2.2 FeatureExtraction

The `FeatureExtraction` step requires a `feature_plan`, a configuration used by the YAAFE library specifying the features to be extracted, and the `training_data_base_path`, the location where the standardized training data is located. The training data is a set of audio files organized into subdirectories (see Figure 5), where the audio files within the same directory belong to the same class. Features are extracted from each of these audio files. The `featureDatasetDestination` CLI parameter can be optionally passed in if we want to save the feature dataset object.

```
> python ./mwpsi/run_classification_workflow.py \
        -c ./tst/jmcModelConfig.json \
        -tasks ExtractFeatures \
        --featureDatasetDestination /<path_to_dir>/dataset/training_data.2018-06-15_17
    .32.23.c1.r44100.s30.noisy.mu.0_sigma.0.01.c0005.n0003.X.y.dataset
```

Listing 3: Execute FeatureExtraction

Listing 4 shows the parameters in the `jmcModelConfig.json` used by the `FeatureExtraction` task.

```
{
  ...
  "extract_features": {
    "feature_plan": "/<path_to_dir>/workspace/featureplan.txt",
    "training_data_base_path": "/<path_to_dir>/workspace/training_data.2018-06-15_17
    .32.23.c1.r44100.s30.noisy.mu.0_sigma.0.01.c0005.n0003",
  },
  ...
}
```

Listing 4: jmcModelConfig.json - Input for Standardization step

Listing 5 shows the contents of the `featureplan.txt` file, which specifies the YAAFE library to extract the Mel-Frequencies Cepstrum Coefficients (MFCC) using a block size of 512, step size between consecutive frames of 256, and to retain 13 cepstral coefficient [16].

```
mfcc: MFCC blockSize=512 stepSize=256 CepsNbCoeffs=13
```

Listing 5: featureplan.txt - Input used by YAAFE for feature extraction

Alternatively, we can override the training data base path through a CLI parameter when running the `FeatureExtraction` step as shown in Listing 6. This will ignore the `training_data_base_path` parameter specified in the `jmcModelConfig.json`

```
> python ./mwpsi/run_classification_workflow.py \
        -c ./tst/jmcModelConfig.json \
        -tasks ExtractFeatures \
        --trainingDataBasePath /<path_to_dir>/workspace/training_data.2018-06-15_17
    .32.23.c1.r44100.s30.noisy.mu.0_sigma.0.01.c0005.n0003/ \
        --featureDatasetDestination /<path_to_dir>/dataset/training_data.2018-06-15_17
    .32.23.c1.r44100.s30.noisy.mu.0_sigma.0.01.c0005.n0003.X.y.dataset
```

Listing 6: Execute FeatureExtraction

4.2.3 ReduceDimensions

The `ReduceDimensions` step requires a CLI parameter, `featureDatasetDestination`, specifying the path to the dataset object, and the .json configuration containing a set of tasks (see Listing 8) to be executed as shown in Listing 7. Currently only PCA is supported but additional dimension reduction techniques can be added. The workflow will create an object for reducing dimensions and if the additional task `get_object_pca` is specified, it will be used to fit the dataset and reduce dimensions.

```
> python ./mwpsi/run_classification_workflow.py \
        -c ./tst/jmcModelConfig.json \
        -tasks ReduceDimensions \
        --featureDatasetDestination /<path_to_dir>/dataset/training_data.2018-06-15_17
    .32.23.c1.r44100.s30.noisy.mu.0_sigma.0.01.c0005.n0003.X.y.dataset
```

Listing 7: Execute ReduceDimensions

```
{
  ...
  "reduce_dimensions": {
      "tasks": ["get_object_pca"]
    },
  ...
}
```

Listing 8: jmcModelConfig.json - Input for ReduceDimensions step

4.2.4 TrainModel

In the `TrainModel` step, we can specify the tasks to run through the `tasks` parameter in the .json configuration.

```
> python ./mwpsi/run_classification_workflow.py \
        -c ./tst/jmcModelConfig.json \
        -tasks TrainModel
```

Listing 9: Execute TrainModel

Listing 10 specifies to use the "SVC" model (`get_object_svc`), a Support Vector Classifier in the scikit-learn library.

```
{
  ...
  "train_model": {
    "tasks": ["get_object_svc"]
  },
  ...
}
```

Listing 10: jmcModelConfig.json - Input for the TrainModel step

4.2.5 AnalyzeModel

The last task, `AnalyzeModel`, uses the configuration in the `ReduceDimensions` and `TrainModel` steps to determine which data transformations and classifier to use for analysis. Listing 11 shows how the

`ReduceDimensions`, `TrainModel`, and `AnalyzeModel` are executed and the corresponding output is saved onto the specified file at the end of the command.

```
> python ./mwpsi/run_classification_workflow.py \
        -c ./tst/jmcModelConfig.json \
        -tasks ReduceDimensions,TrainModel,AnalyzeModel \
        --featureDatasetDestination /<path_to_dir>/workspace/dataset/training_data
    .2017-11-23_21.02.20.c1.r44100.s30.noisy.mu.0_sigma.0.01.c0005.s0125.X.y.labels.
    dataset \
        | tee /<path_to_dir>/workspace/dataset/training_data.2017-11-23_21.02.20.c1.
    r44100.s30.noisy.mu.0_sigma.0.01.c0005.s0125.X.y.labels.output
```

Listing 11: Execute AnalyzeModel

Listing 12 specifies to use `cross_validation` in the `jmcModelConfig.json` configuration for analyzing the model.

```
{
  ...
  "analyze_model":{
    "task" : ["cross_validation"]
  }
  ...
}
```

Listing 12: jmcModelConfig.json - Input for AnalyzeModel step (uses configurations from ReduceDimensions and TrainModel steps)

Currently, the tuned parameters for cross validation are built into the code. We perform hyperparameter optimization using cross validation with GridSearch to compare different parameter combinations and determine the best set of parameters to use. Figure 13 shows the parameters used for analyzing the model. PCA is used to transform the data using 2-21 principal components. A SupportVectorClassifier (SVC) is used with a radial basis function (RBF) kernel and the specified C and γ parameters. Similar analysis is done using a linear kernel. We specify the use of *accuracy*, *precision_macro*, *recall_macro*, *f1_macro*, and *f1_weighted* scoring functions as model evaluation metrics for determining quality of the classifier.

```
tuned_parameters = [{'svc__kernel': ['rbf'],
                     'svc__gamma': [0.001, 0.01, 0.1, 1.0],
                     'svc__C': [0.5, 1, 1.5, 2],
                     'pca__n_components':np.arange(20)+2},
                    {'svc__kernel': ['linear'],
                     'svc__gamma': [0.001, 0.01, 0.1, 1.0],
                     'svc__C': [0.5, 1, 1.5, 2],
                     'pca__n_components': np.arange(20)+2}]
scoring = ["accuracy", "precision_macro", "recall_macro", "f1_macro", "f1_weighted"]
```

Listing 13: Parameters used for performing hyperparameter optimization

4.3 Experiments

Three types of experiments were conducted, each used to focus on a different facet of data and model evaluation.

4.3.1 Data Dimensionality

In the first experiment we used the original dataset consisting of 5 classes, each with 5 samples. This results in a total of 25 data samples. After extracting MFCC features, each sample was 67184-dimensional (67184 feature values). Twenty-five is not sufficient data to generalize and analyze the behavior of the model. However, this experiment is useful for analyzing the structure of the data and the reduction of dimensions. We used PCA to reduce the dimension space using 25 principal components.

4.3.2 Model Evaluation Metrics

In the second experiment, we used the original dataset as a seed dataset to generate 25 noisy samples from each seed sample. Random noise sampled from a standard distribution with $\sigma = 0.1$ and $\mu = 0$ was added to a seed sample. This generated 125 noisy samples for each of the 5 class resulting in a training set of size 625 (625 training samples with dimension 67184).

$$\frac{5 \text{ seed samples}}{\text{class}} * \frac{25 \text{ noisy samples}}{\text{seed sample}} = \frac{125 \text{ noisy samples}}{\text{class}} \tag{2}$$

In this experiment, we trained an SVM model and conducted hyperparameter optimization of the principal components, kernel function, γ, and C parameters using grid search with $k = 5$ cross validation on 80% of the training set. We used the following optimization parameters:

Table 1: **Experiment 2: Hyperparameter optimization for metric evaluation**

γ	0.001, 0.01, 0.1, 1
C	0.5, 1, 1.5, 2
principal components	2, 3, ..., 20, 21

We used small values for γ in powers of 10 to see the effect of γ at different magnitudes. We used small values for C to avoid precision issues from occurring when using small γ and large C values that can lead to the γ parameter being overpowered by the magnitude of C. We used 20 principal components because after reducing the data, we saw that using more than 20 features did not provide additional distinguishing features. The range is 2 to 21 because we started with 2 dimensions given that 1 dimension is not enough information to distinguish between the different classes. In this section we focused on understanding the different metrics by exploring a small range of hyperparameters. The main purpose for this experiment was to analyze the experiment setup with different model evaluation metrics, namely, *accuracy*, *precision_ macro*, *recall_ macro*, *f1_ macro*, *f1_ weighted* as defined in the equations that follow and for which $F_1 = F_{\beta=1}$. Given that

y is the set of predicted $(sample, label)$ pairs
$\hat{y}$ the set of true $(sample, label)$ pairs
L the set of labels
tp are true positives
fp are false positives
fn are false negatives
$P(A, B) := \frac{|A \cap B|}{|A|}$
$R(A, B) := \frac{|A \cap B|}{|B|}$(Conventions vary on handling$B = \emptyset$; this implementation uses $R(A, B) := 0$, and similar for P.)

the metrics are defined as follow [21]:

$$\begin{aligned} \texttt{accuracy}\,(y, \hat{y}) &= \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i) \\ precision &= \frac{tp}{tp + fp} \\ recall &= \frac{tp}{tp + fn} \\ F_\beta &= (1 + \beta^2)\frac{\text{precision} \times \text{recall}}{\beta^2 \text{precision} + \text{recall}} \end{aligned} \tag{3}$$

The macro and weighted averaging are defined as:

$$F_\beta(A, B) := \left(1 + \beta^2\right) \frac{P(A, B) \times R(A, B)}{\beta^2 P(A, B) + R(A, B)}$$

$$\begin{aligned} \texttt{macro} &: \frac{1}{|L|} \sum_{l \in L} F_\beta(y_l, \hat{y}_l) \\ \texttt{weighted} &: \frac{1}{\sum_{l \in L} |\hat{y}_l|} \sum_{l \in L} |\hat{y}_l|\, F_\beta(y_l, \hat{y}_l) \end{aligned} \tag{4}$$

After obtaining the optimal model, h^*, as evaluated by the specified metric, we then evaluated on the remaining 20% of the data and calculated precision and recall numbers on that dataset.

4.3.3 Hyperparameter Optimization

In the third experiment, the setup was similar to that used in the second experiment with a wider range of hyperparameters and focusing on the behavior of the γ and C parameters. We used the parameters defined in Table 2.

Table 2: **Experiment 3: Hyperparameter optimization**

C	0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000
γ	0.0001, 0.001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000
principal components	2, 3, ..., 20, 21
kernel	linear, rbf

5 Analysis of Results

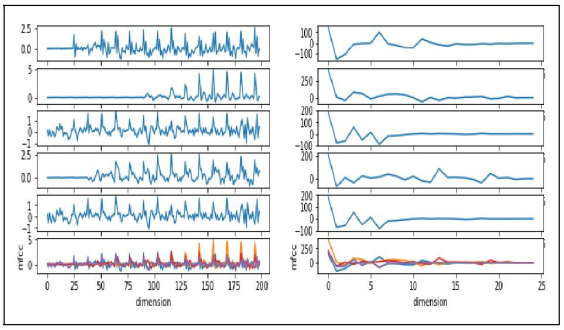
In the following sections, we provide an analysis on the experiments conducted for analyzing the data, evaluating the model, and model hyperparameter optimization.

5.1 Data Dimensionality

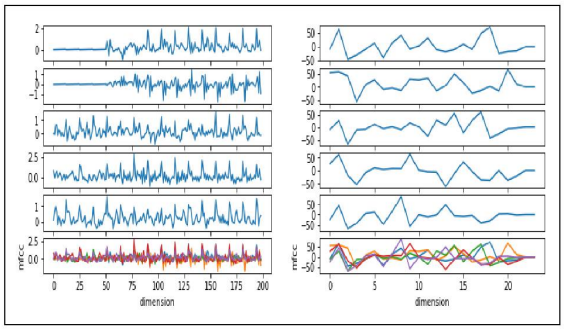
We extracted features from the audio data samples using 13 Mel Frequecy Cepstral Coeffients (MFCC) which resulted in 67184 features for each standardized data sample.

Due to limitations in plotting high dimensional data, we plotted each feature vector independently instead, where the horizontal axis is the dimension and the vertical axis is the MFCC value. Figure 6 (a)-(e) shows the 5 data samples in each class. The graphs on the left in each class grouping show the feature vectors with the first 200 features (out of a total of 67184). Clearly, we are able to see recurring patterns in each feature vector and thus, it's unnecessary to plot all 67184 features. Note that the first 13 features are features from the first frame, the second set of 13 features are features from the second frame, etc. From this we expect roughly every set of 13 features to be similar in pattern if there is a consistency in the audio recording. After applying PCA and mapping the feature vectors (left images in each set of class) to 25 principal components, the resulting vectors are the simplified feature vectors shown on the right in each set of class. We looked at 25 components and saw that beyond 20 dimensions, there aren't many differentiating characteristics between the samples. Thus, it is unnecessary to train on data beyond 20 dimensions as it would only add computational cost. The two bottom (left and right) graphs in each class grouping is an aggregate of the 5 feature vectors above them. This helps us see recurring patterns among the samples within each class. The reduced vectors highlight these patterns more clearly. For example, the curves in class 0 (Figure 6(a)) and class 4 (Figure 6(e)) are clearly distinguishable. The curve in class 0 starts out positive with a steep negative slope whereas class 4 starts out negative with a steep positive slope. Class 1 also looks like it has a consistent pattern at the beginning across all samples. However, class 2 and class 3 seem harder to distinguish. In Figure 6(f)(left), we show all the reduced

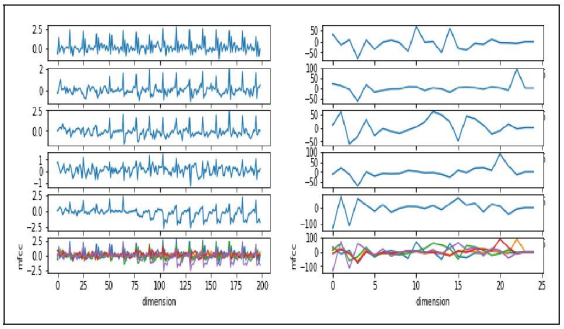
feature vectors with the first 2 features in $\mathbb{R}^2$ where we can see that green, red, and purple colored classes seem intertwined. In Figure 6(f)(right), we show the same vectors with 3 features in $\mathbb{R}^3$, where we can more clearly see separation between the green, red, and purple colored classes.



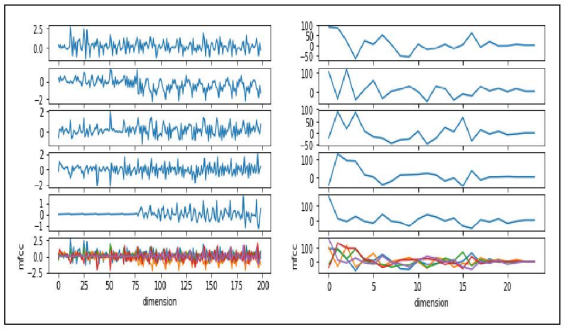
(a) class 0 - bald eagle



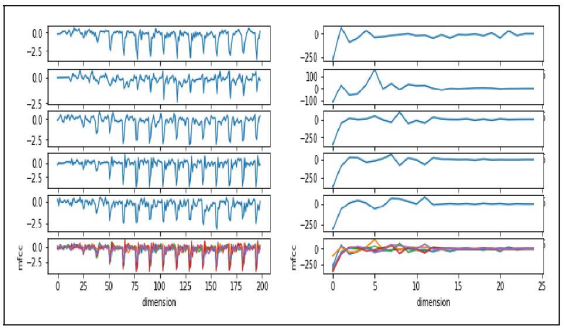
(b) class 1 - black caracara



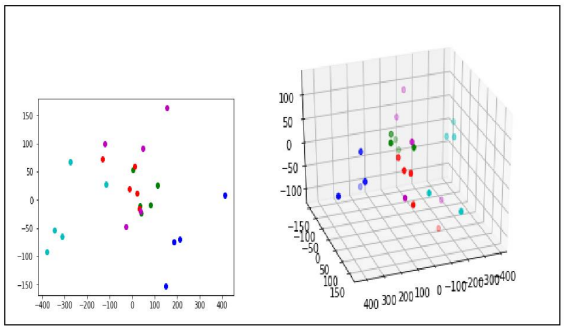
(c) class 2 - common wood pigeon



(d) class 3 - fox sparrow



(e) class 4 - great tinamou



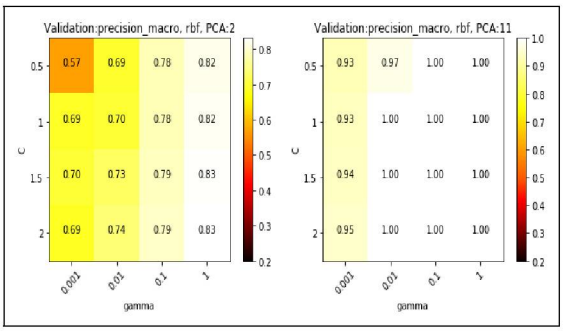
(f) Left: 25 feature vectors with first 2 dimensions (5 samples per class), Right: feature vector with first 3 dimensions

Figure 6: The graphs show the feature vector and the corresponding transformation after applying PCA. Left: Feature vector containing the first 200 MFCC features, Right: Transformed feature vector after applying PCA transformation with 25 principal components. See Appendix A for larger images

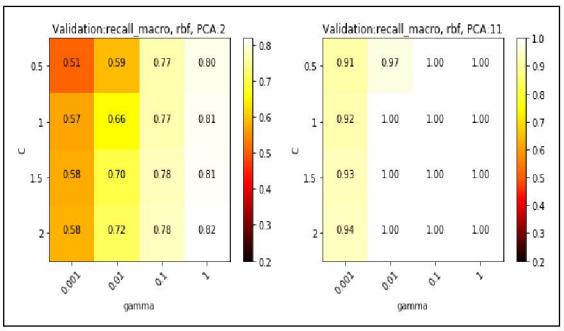
5.2 Model evaluation metrics

The models were trained using $k = 5$ cross-validation on 80% of the dataset (500 random samples with 104 from class 0, 97 from class 1, 93 from class 2, 105 from class 3, and 101 from class 5). Figure 7 (a)-(e) show the mean training model evaluation scores for metrics: *precision_ macro*, *recall_ macro*,

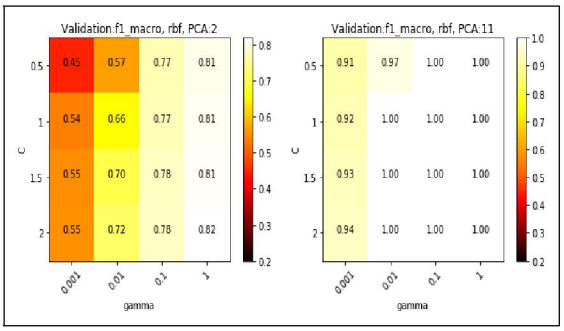
f1_ macro, *f1_ weighted*, and *accuracy*. In each graph group, we show the scores from training using 2 and 11 principal components in the left and right graph, respectively. The graphs clearly show that a higher number of components yields a better trained model. Precision looks at the accuracy of each class while recall looks at the number of correct samples covered in each class. This is why the scores for *precision_ macro*, *recall_ macro* and *accuracy* have similar results. Note that *precision_ macro* and *accuracy* are similar metrics. However, *accuracy* treats all samples equally and biased towards classes with a larger number of training samples. The *f1_ macro* and *f1_ weighted* metrics generated similar results with slight higher scores for *f1_ weighted*. For example, *f1_ weighted* has higher scores for the model with parameters $\gamma = 0.001$, $C = 0.5$, principal components = 2 (0.45 for *f1_ macro* and 0.46 for *f1_ weighted*) and the model with parameters $\gamma = 0.001$, $C = 0.5$, principal components = 11 (0.91 for *f1_ macro* and 0.92 for *f1_ weighted*). The F1-score incorporates both precision and recall numbers and while precision and recall (Figure 7 (a)-(b)) provide good scores, these metrics work well with training data with relatively equal class distribution, which is our case. Therefore, we use the *f1_ weighted* metric for evaluating the models.



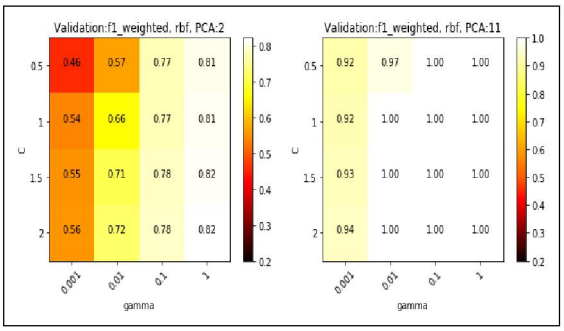
(a) *precision_ macro*



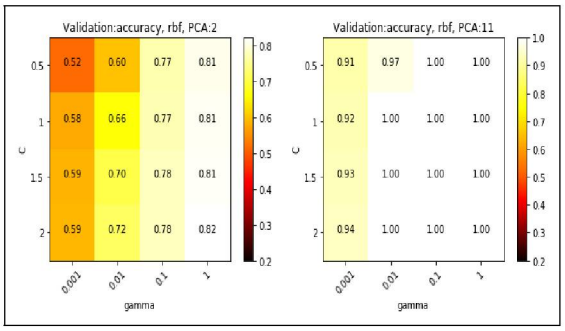
(b) *recall_ macro*



(c) *f1_ macro*



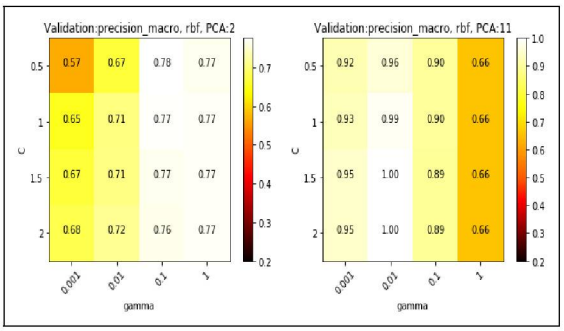
(d) *f1_ weighted*



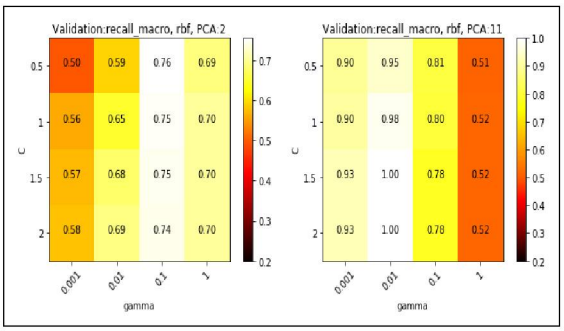
(e) *accuracy*

Figure 7: The graphs show the model's mean training score for the given evaluation metric. *Left*: model scores using 2 principal components, *Right*: model scores using 11 principal components. See Appendix B for larger images.

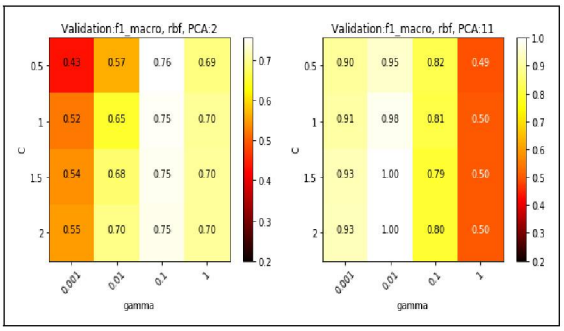
In Figure 7, we show the mean training scores during cross validation. In Figure 8 we show the mean test scores during cross validation. These scores are evaluated on the kth subset of samples that was left out during training with $k-1$ subsets. Note that the scores are slightly lower for models with 2 principal components. This shows the importance on testing on data not yet "seen" by the model. Note that in this case, we have different results for models with 11 principal components. Namely, the model performs worse with a small γ value and a small number of components as well as with a large γ and large number of components. Decent values for C appear to be numbers with a magnitude of 1. The fact that larger γ values are better with low dimensional dataset and smaller γ values are better for higher dimensional datasets illustrate the importance of hyperparameter optimization.



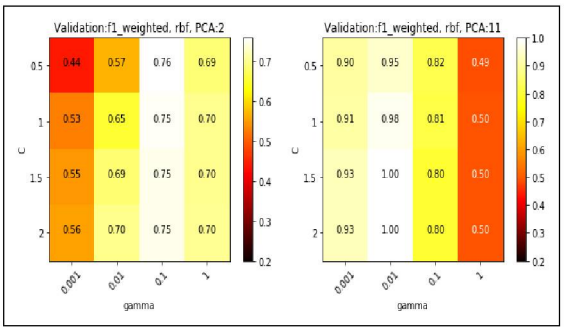
(a) *precision_ macro*



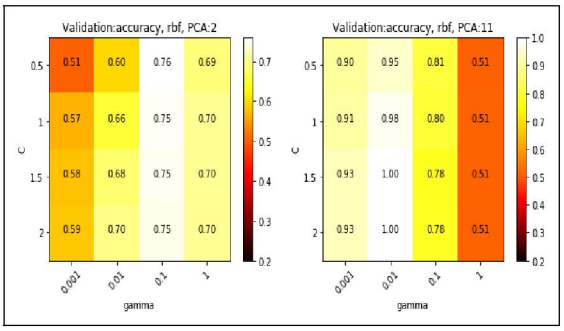
(b) *recall_ macro*



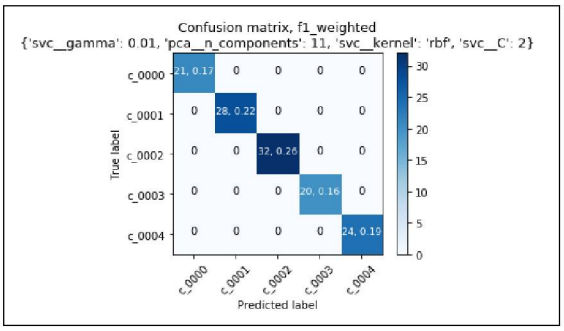
(c) *f1_ macro*



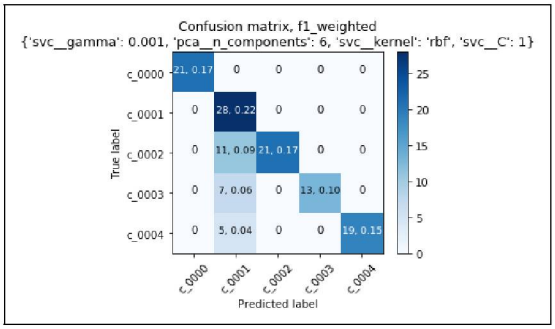
(d) *f1_ weighted*



(e) *accuracy*



(f) classification best model evaluated with *f1_ weighted* metric



(g) Results from a poorly performing classifier

	precision	recall	f1-score	support
0	1.00	1.00	1.00	21
1	0.55	1.00	0.71	28
2	1.00	0.66	0.79	32
3	1.00	0.65	0.79	20
4	1.00	0.79	0.88	24
avg / total	0.90	0.82	0.83	125

(h) Metric scores of the poorly performing classifier in Figure 8(g)

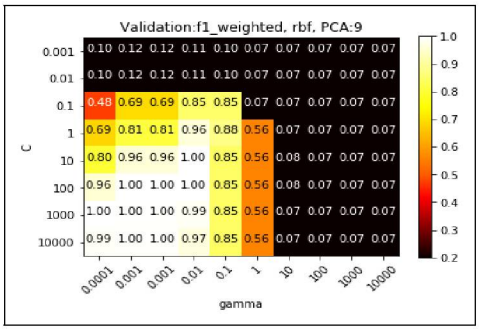
Figure 8: The graphs show the model's mean test score for the given evaluation metric. *Left*: model scores using 2 principal components, *Right*: model scores using 11 principal components. See Appendix C for larger images.

In this experiment, the optimal model, h^*, as evaluated by all metrics was the model with 11 principal components, $C = 2$, rbf kernel, and $\gamma = 0.01$. Figure 8(f) shows the confusion matrix of the remaining

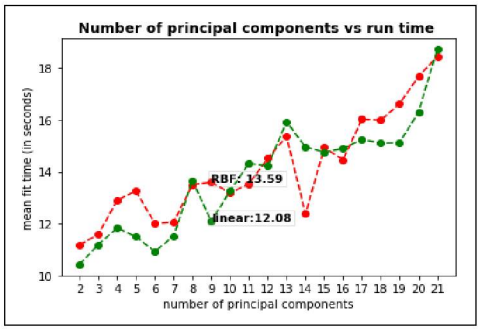
20% (125 samples) of the original training dataset. Note that all samples were correctly predicted. Compare with the results from the poorly performing classifier in Figure 8(g) where 23 (11 from class 2, 7 from class 3, and 5 from class 4) of the samples were incorrectly classified as belonging to class 1. In Figure 8(h), we show the *precision*, *recall*, and *f1* scores of the poorly performing classifier.

5.3 Hyperparameter optimization

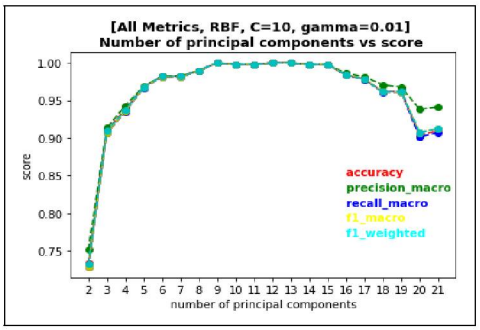
We used the *f1_ weighted* metric, to optimize for the number of principal components, kernel function, γ, and C parameters. In addition, we investigated the fit time as this is an important metric especially as we increase our dataset size and dimensionality.



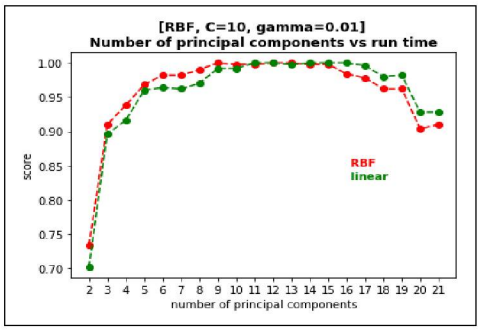
(a) *f1_ weighted* scores



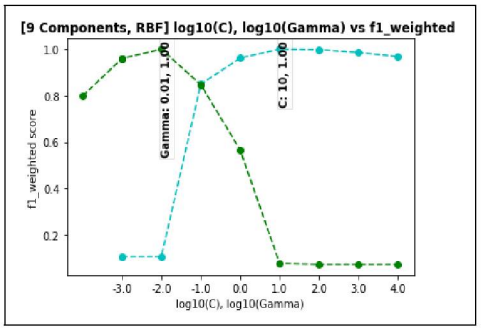
(b) Run time results



(c) Scoring for model with RBF kernel, $C = 10$, and $\gamma = 0.01$ for varying number of principal components



(d) kernel function comparisons (rbf, linear)



(e) C and γ parameters

Figure 9: The figures show the different facets of the model's performance based on different hyperparameters. See Appendix D for larger images.

Figure 9(a) shows the mean *f1_weighted* test scores during cross validation with a larger γ and C range (as compared to section 5.2). The optimal model, h^*, was the one with 9 principal ocmponents, rbf kernel, $C = 10$, and $\gamma = 0.01$. Note that models with other parameter values also generated high scores but the numbers for C and γ are either too large or too small. Having such extreme values can lead to overflows and inaccurate results.

In Figure 9(b), we show that the fit runtime for the optimal model grows linearly as the number of components increases. Hence, we must be judicious when considering adding more dimensions to our dataset.

In Figure 9(c), we overlay the score results of the optimal model using different evaluation metrics and see that all metrics return similar results.

In Figure 9(d), we plot the model evaluation score for models with RBF and linear kernels with varying number of principal components. The models' performances are comparable with RBF having slightly better (+3%) performance when using 2-9 components. The performance then plateaus for both kernels up until 15 components and then begin to decrease for larger number of components. We clearly see higher performance for models with the RBF kernel.

Figure 9(e) shows the *f1_weighted* scores for a model with an RBF kernel, 9 principal components, and different C and γ parameters. We scale the x-axis to $log_{10}(C)$ and $log_{10}(\gamma)$ to make it easier to compare the score results. The graph shows that the model performance behaves opposite as we vary the magnitude of these two parameters. Smaller γ values are better while slightly larger C values are better. Note that C is the regularization parameter that is used to determine the weight of the cost associated to incorrect classifications. A larger C value will tend to overfitting. The γ parameter is used to determine how smoothly the samples vary. The optimal model had $\gamma = 0.01$, a small value, indicating that the model leans towards the high variance side (overfitting). Though the model performs "well" on the remaining 20% of the dataset, the model may have overfitted the data. The reason why the model may have selected a small value for γ may be to compensated for the fact that 3 of the classes are very close together (the features are very similar). The model correctly classified the 20% of the dataset nonetheless.

6 Conclusion

In this study, we developed a framework for identifying wildlife sounds. From an original dataset of 5 classes each with 5 unique samples, we generated a dataset of 625 noisy training samples (125 samples

per class). The components in the classification workflow were built in a modular way to allow for rapid development, data scalability, and integration flexibility. As a result, each step can be ran independently and improved without affecting other components. We analyzed the data, created an SVM model, evaluated the model using different metrics and performed hyperparameter optimization to generate the optimal model. Results showed a linear runtime increase as the number of principal components increased. The RBF kernel performed $\approx 3\%$ better than the linear kernel, and overall model performance peaked at $\gamma = 0.01$ and $C = 10$ with an $f1\_weighted$ score of 1. Note that the quality and performance of the model is dependent on the quality and diversity of the data. The results in this study were conducted in a controlled setting but in practice, we may need to add additional data transformation pre-processing steps such as noise reduction. Other dimensions need to be considered for identifying bird species on the Santa Rosa Island, such as the bird age and the time of day. In 2015, during the CI Rainbow Research Project trip to the Santa Rosa Island, bird surveyors indicated that birds may generate different sounds depending on their age and may sing hundreds of different songs, making it difficult identify a bird type based on only one type of song. The model created with this framework is a baseline for identifying wildlife on the Santa Rosa Island. The classifier will improve over time as we obtain more data. Given the different types of wildlife, we can utilize different models and create more sophisticated classifiers, such as ensembles and decision trees, to improve classification as we introduce new wildlife samples to the model. This automatic identification process will help us monitor wildlife and aid in the effort of preserving flora and fauna.

7 Future Work

Future work on the framework can be categorized into productionization, deployment, integration, and continuous improvement. While this library can be used for different applications, the current state is not production ready. One of the current limitations is the manual setup of dependent libraries for running audio feature extraction and Python libraries. Productionizing this module entails packaging the module into a standardized development unit such as a Docker container that would facilitate future development and deployment.

Once productionized, we need to integrate and deploy the library to the existing CI Rainbow infrastructure. In Figure 10, we provide a visualization of the `mwlpsi-lib` component (part of the Machine Learning Module colored in orange) that we developed incorporated into the CI Rainbow infrastructure. In the figure, we show 4 workflows: (1) R1: The wildlife sound gets recorded by the sensor at the Santa Rosa Island. R2: The Machine Learning Module, deployed to the sensor or another computing component at the island, uses the `mwlpsi-lib` to process the recording and classify the sound (Fox sparrow

wildlife). R3: The data gets sent to the mainland, where the CI Rainbow Server runs. R4: The data is saved onto the corresponding databases. (2) Blue workflow: A user accesses the Data Analysis System through the CI Rainbow Web Server to create Jupyter Notebooks, where they can access data from existing databases and use the Machine Learning Module to create new models. The user can deploy new models through the CI Rainbow Server from the mainland to the Island. Once completed with analysis, the user can save their notebook. (3) Green workflow: The user can use the Wildlife Management component within the CI Rainbow Web Server to manage wildlife data. They can introduce new species and add relevant information. (4) Orange workflow: The user can use the Visualization component through the CI Rainbow Web Server to map out the population of different wildlife. The Data Analysis System, Wildlife Management, and Visualization components are being developed by other students and will need to be updated to interact with the Machine Learning Module.

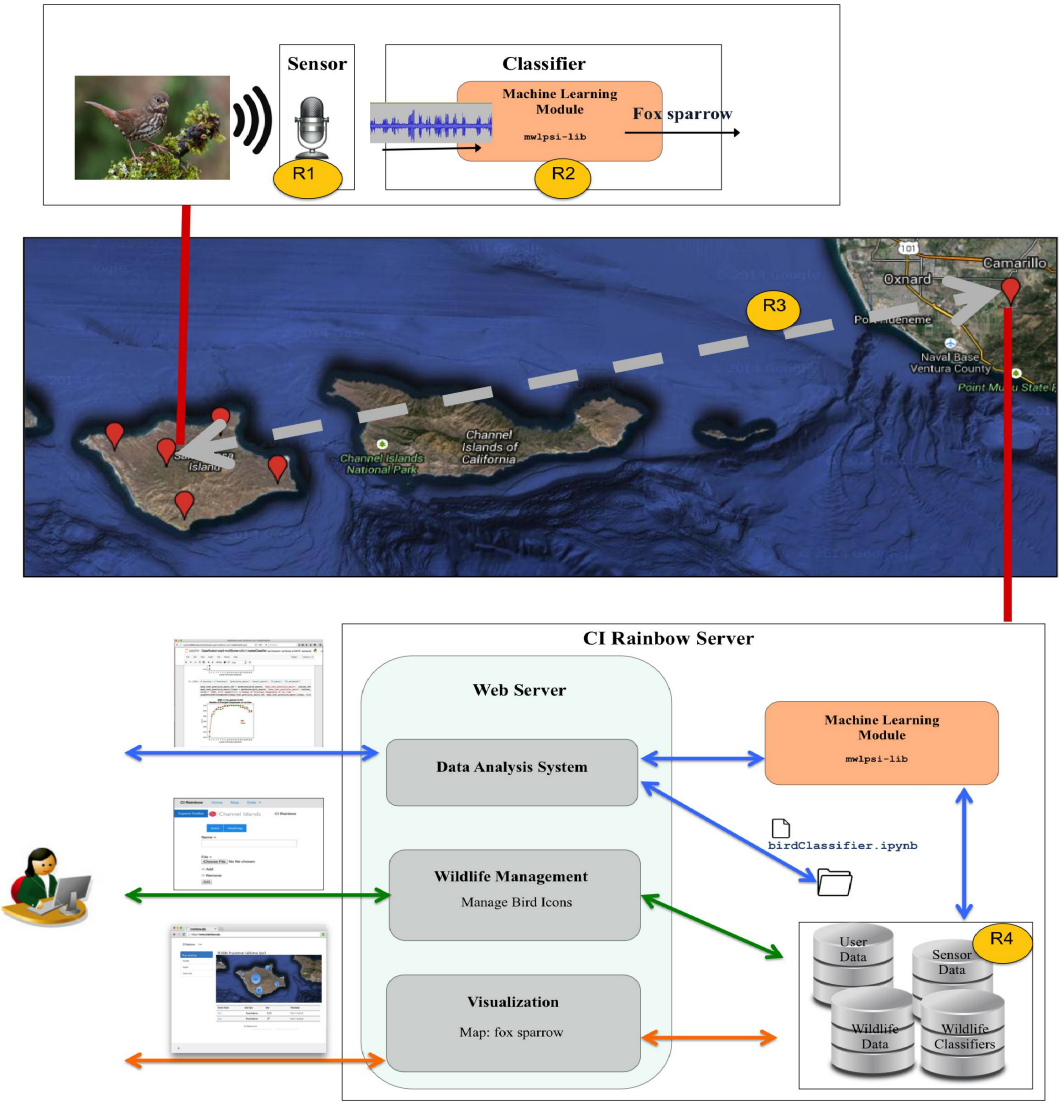


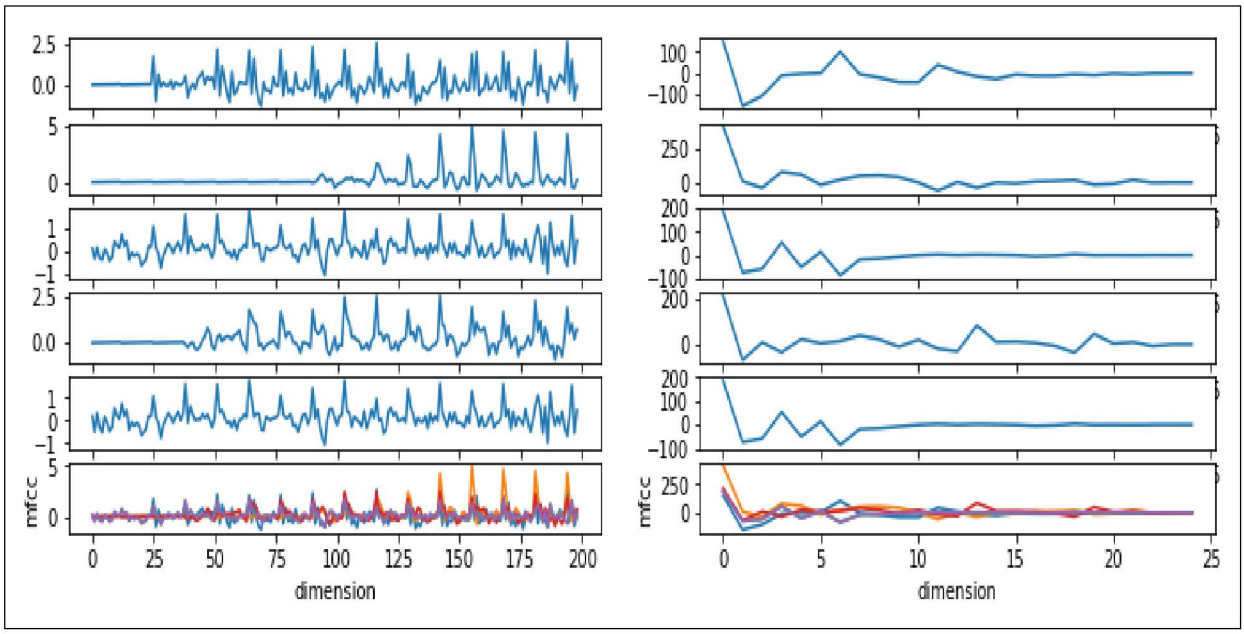
Figure 10: CI Rainbow Infrastructure

Software improvements to the `mwlpsi-lib` component includes adding functionality to support other data types (i.e. data from sensor nodes recording weather data, images, video, etc.), adding support

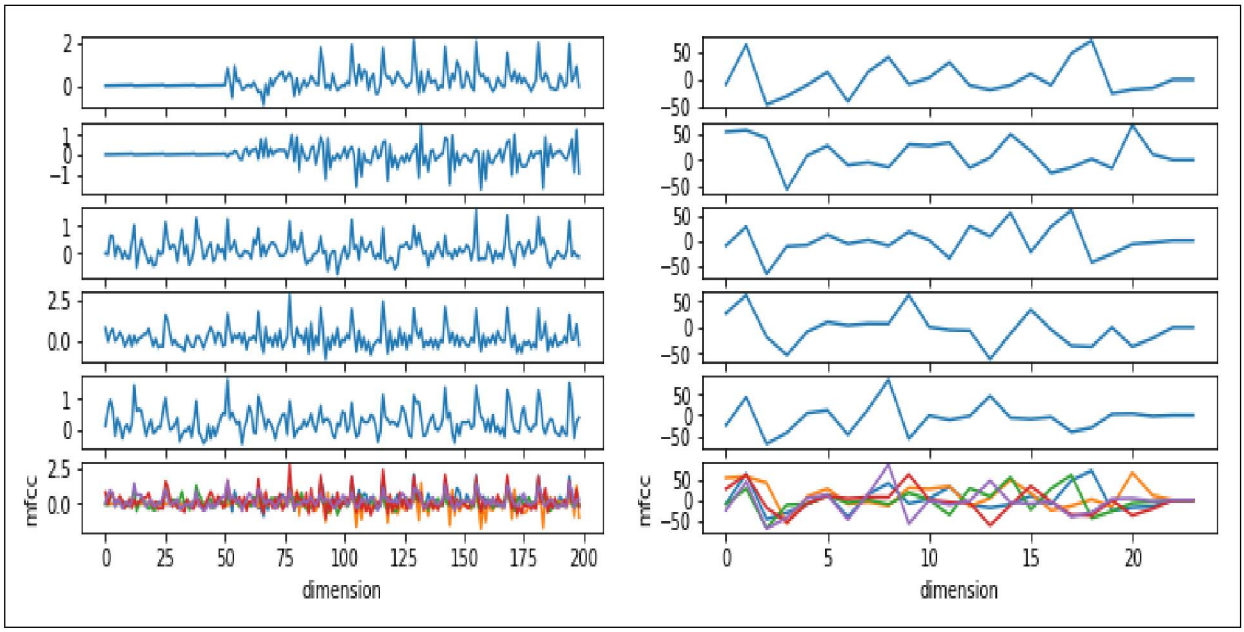
for other feature extraction techniques, adding additional classifiers, optimizing, and refactoring and generalizing code to name a few. Given the large amount of data that we may want to process, we can make improvements to support processing the data in a scalable and distributed manner for faster computation. While this project creates a framework for identifying wildlife sounds for the purpose of modeling wildlife population, it is also a framework for applying other data mining techniques that can be integrated in many other applications.

Appendix A Experiment 1 - Data Dimensionality

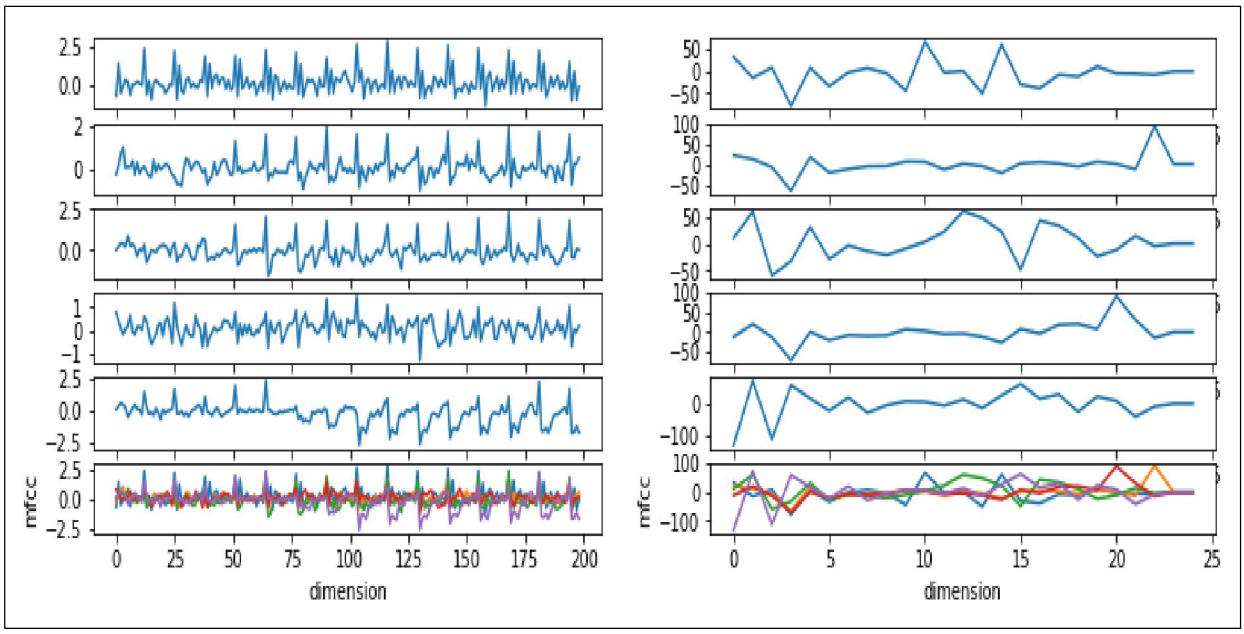
The graphs show the feature vector and the corresponding transformation after applying PCA. Left: Feature vector containing the first 200 MFCC features, Right: Transformed feature vector after applying PCA transformation with 25 principal components



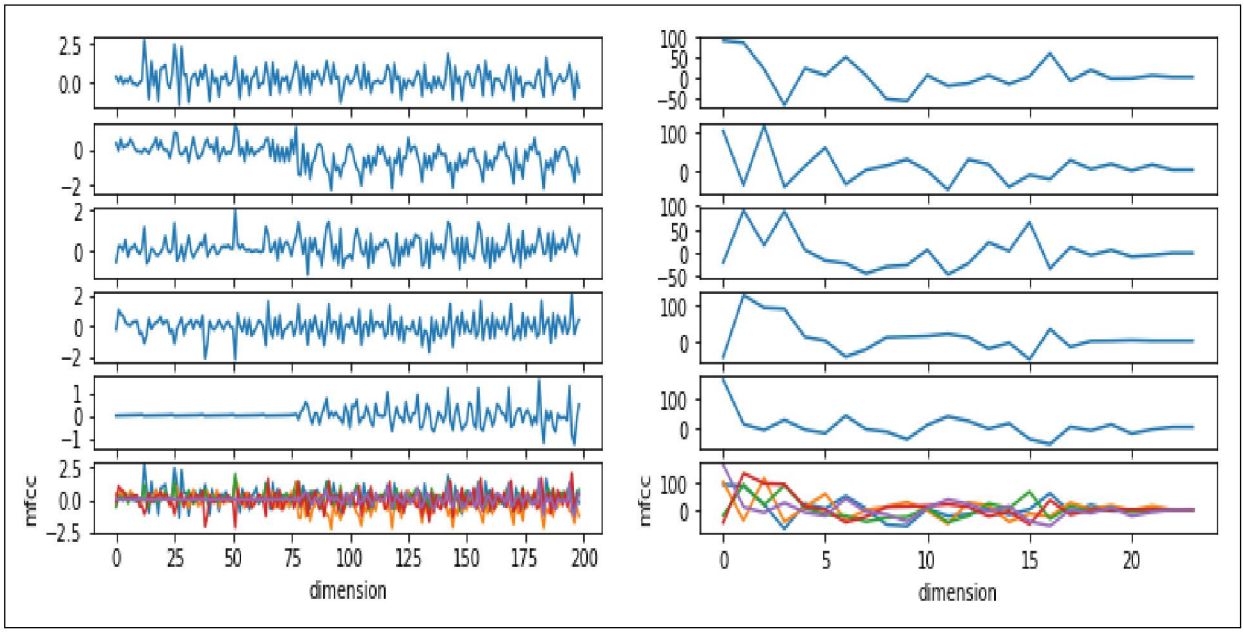
From Fig 6 (a) class 0 - bald eagle



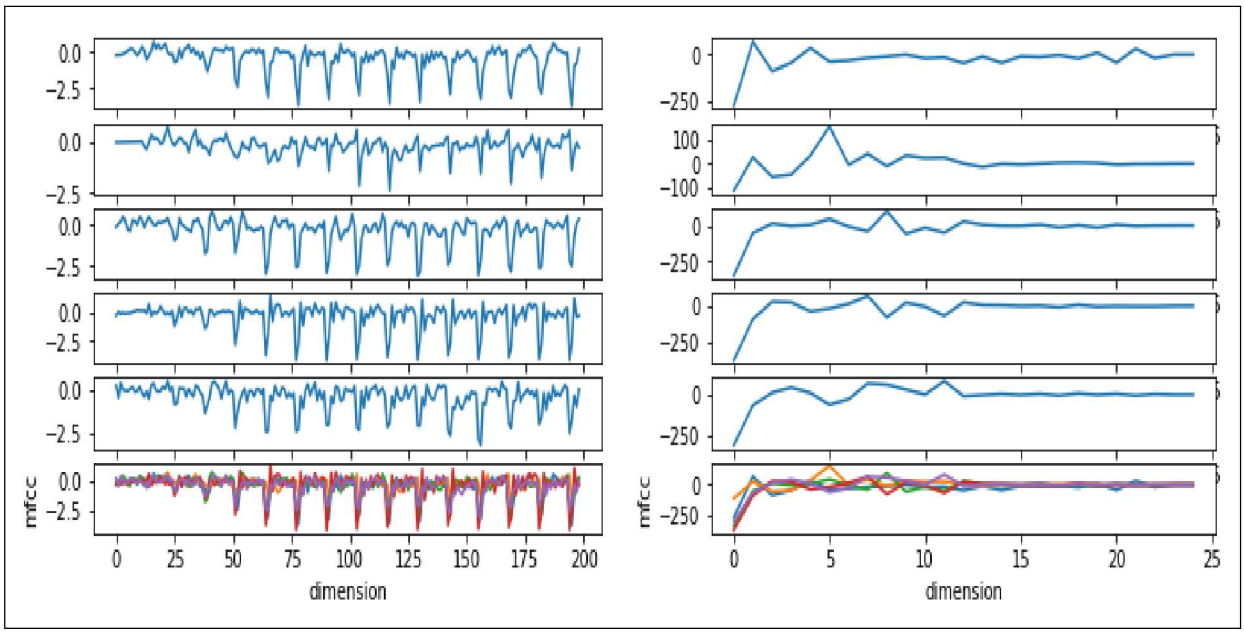
From 6 (b) class 1 - black caracara



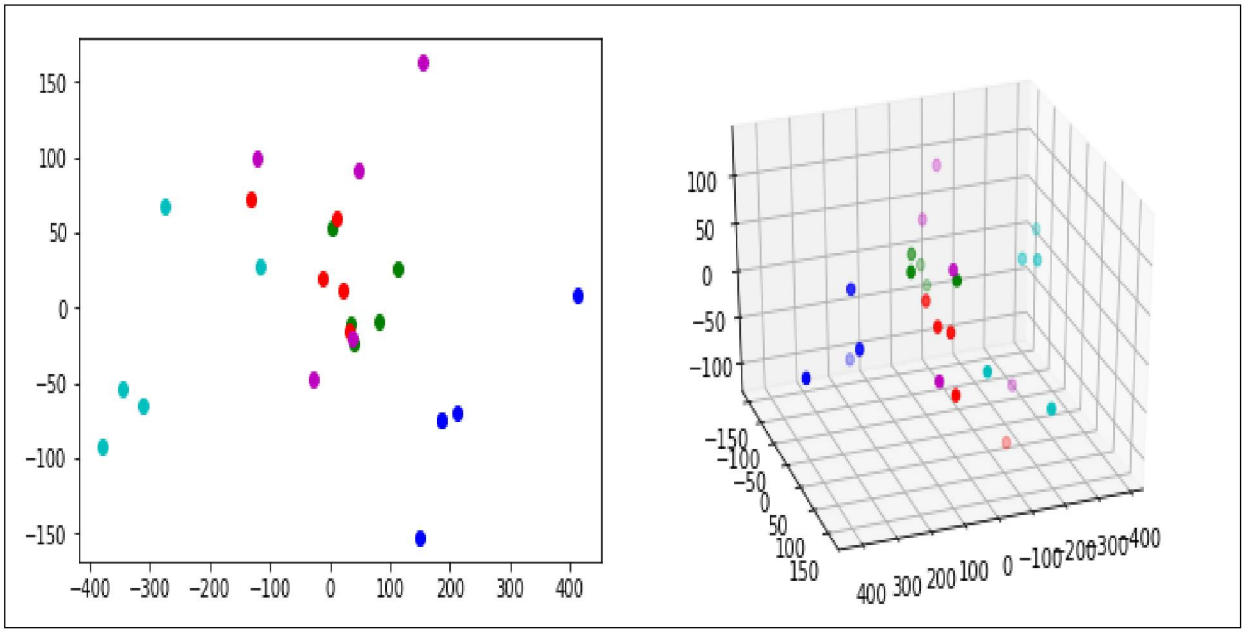
From 6 (c) class 2 - common wood pigeon



From 6 (d) class 3 - fox sparrow



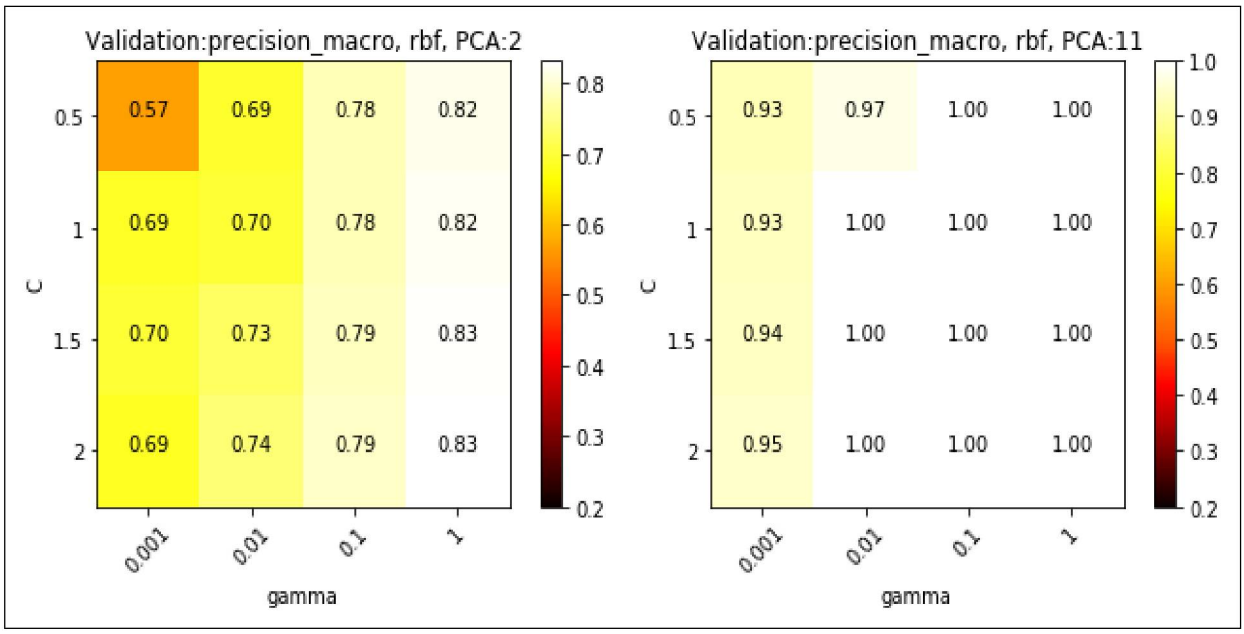
From 6 (e) class 4 - great tinamou



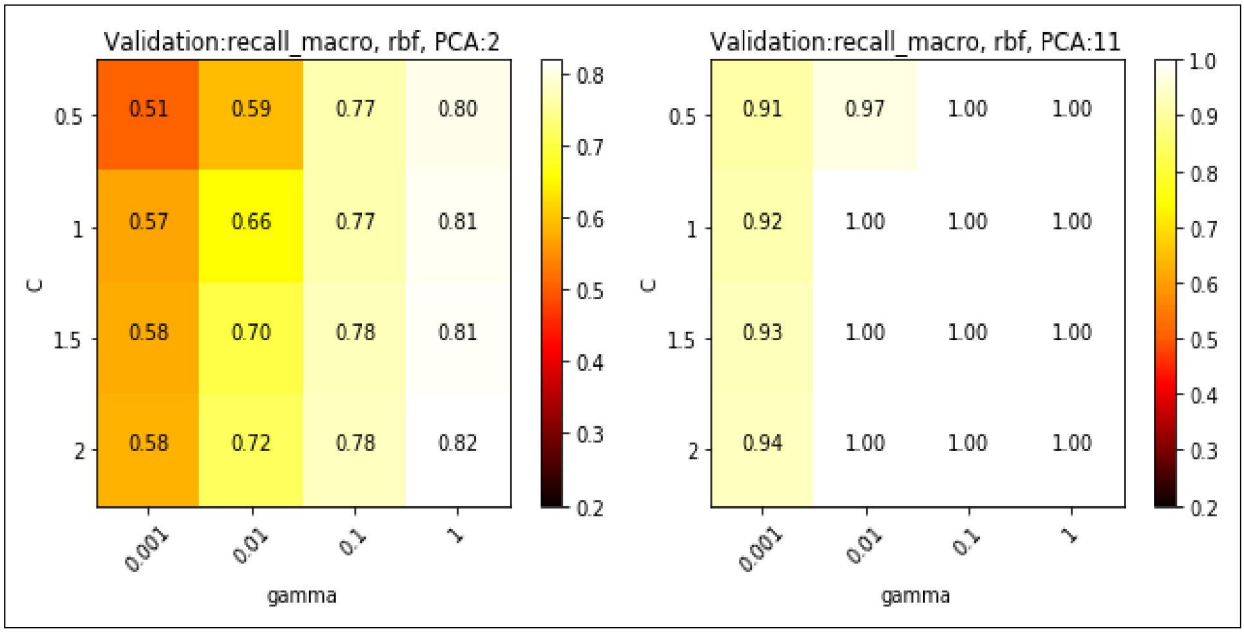
From 6 (f) Left: 25 feature vectors with first 2 dimensions (5 samples per class), Right: feature vector with first 3 dimensions

Appendix B Experiment 2 - Model evaluation metrics: training scores

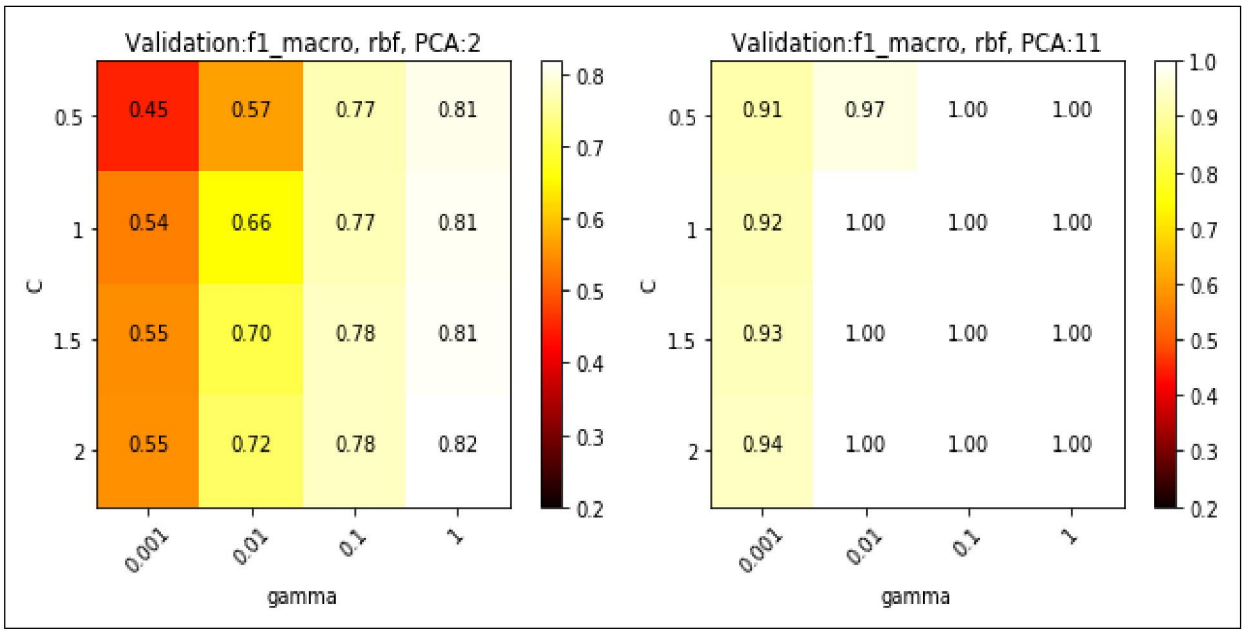
The graphs show the model's mean training score for the given evaluation metric.



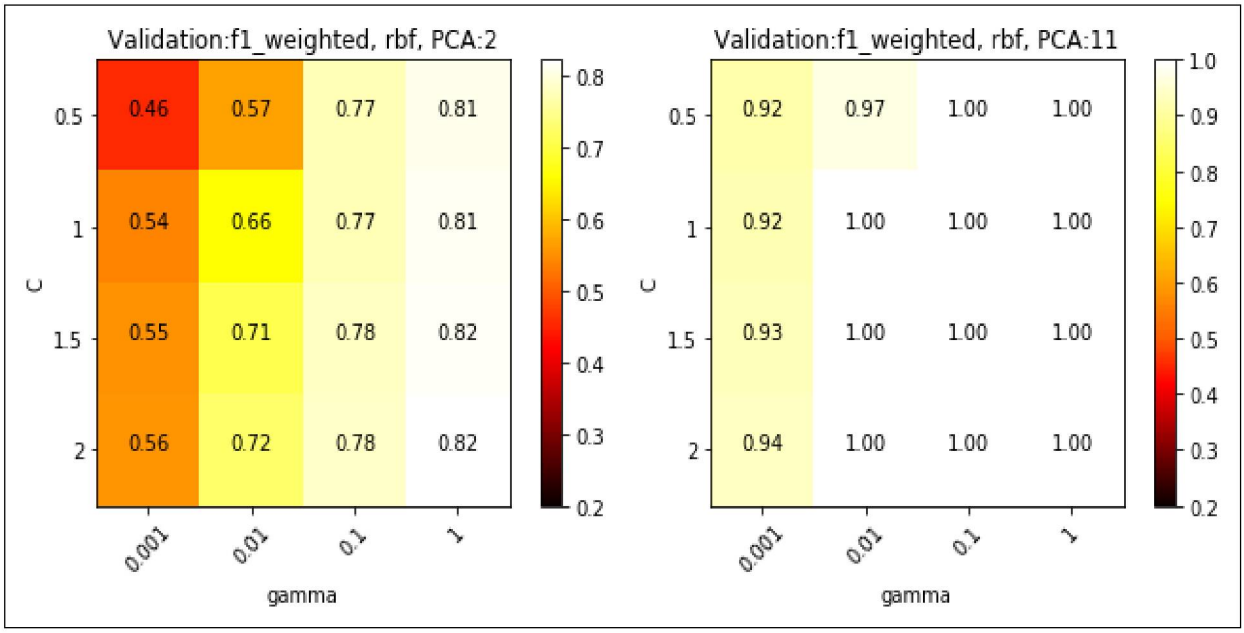
From Fig 7, *precision_ macro*



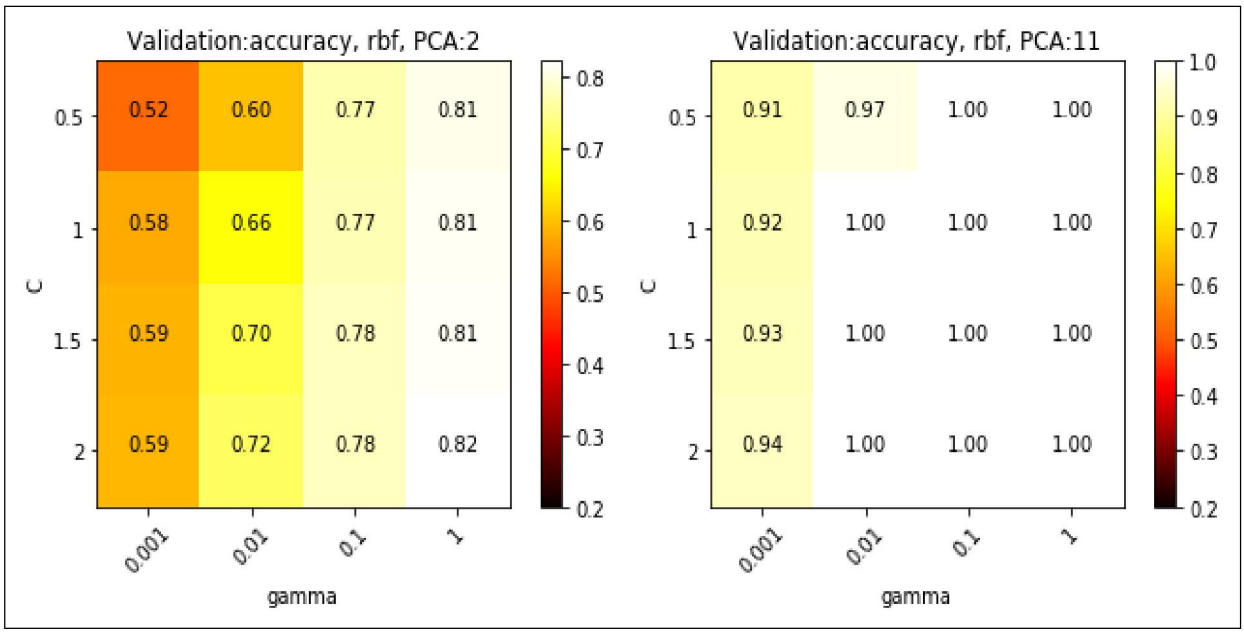
From Fig 7, *recall_ macro*



From Fig 7, *f1_ macro*



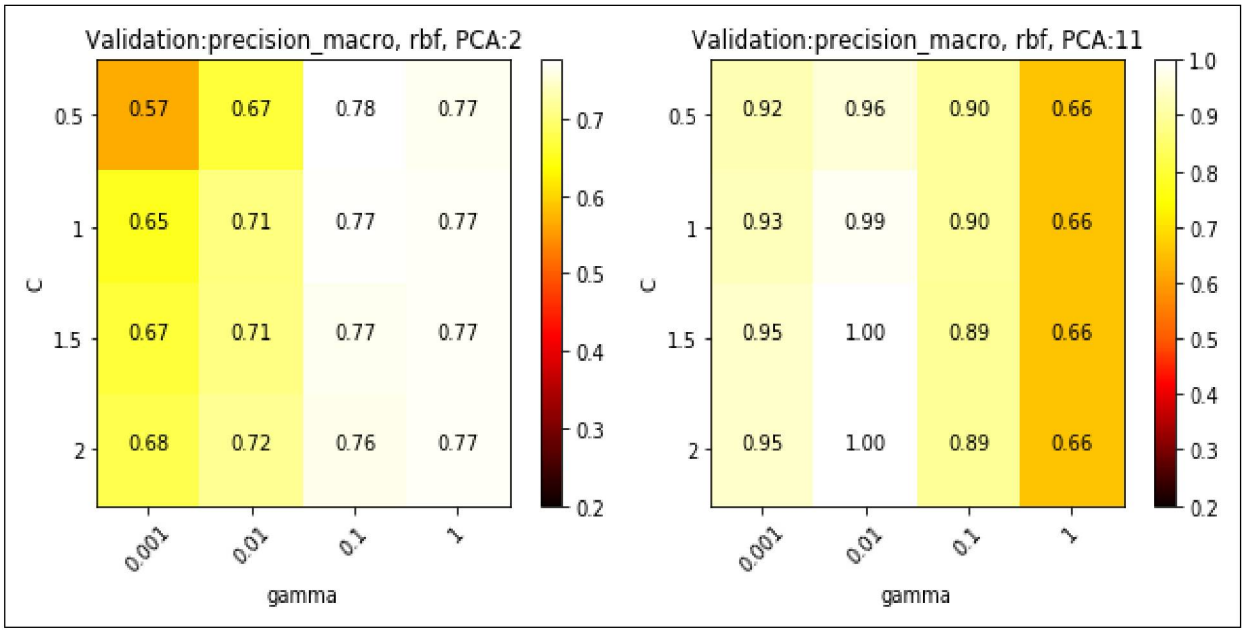
From Fig 7, *f1_ weighted*



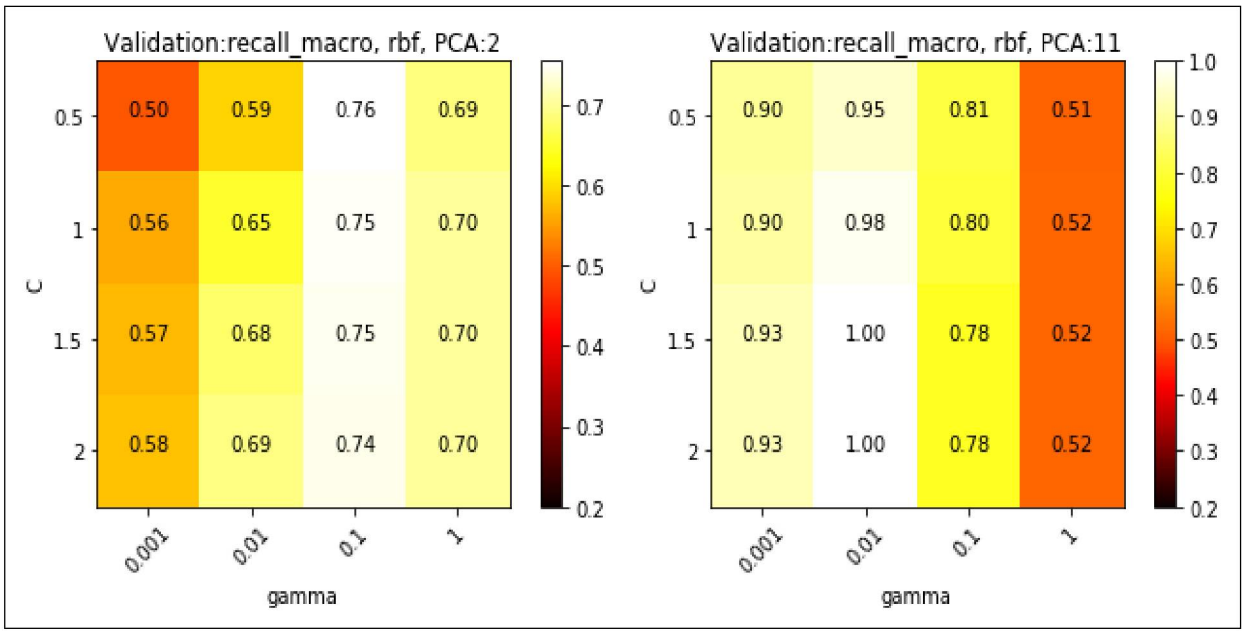
From Fig 7, *accuracy*

Appendix C Experiment 2 - Model evaluation metrics: test scores

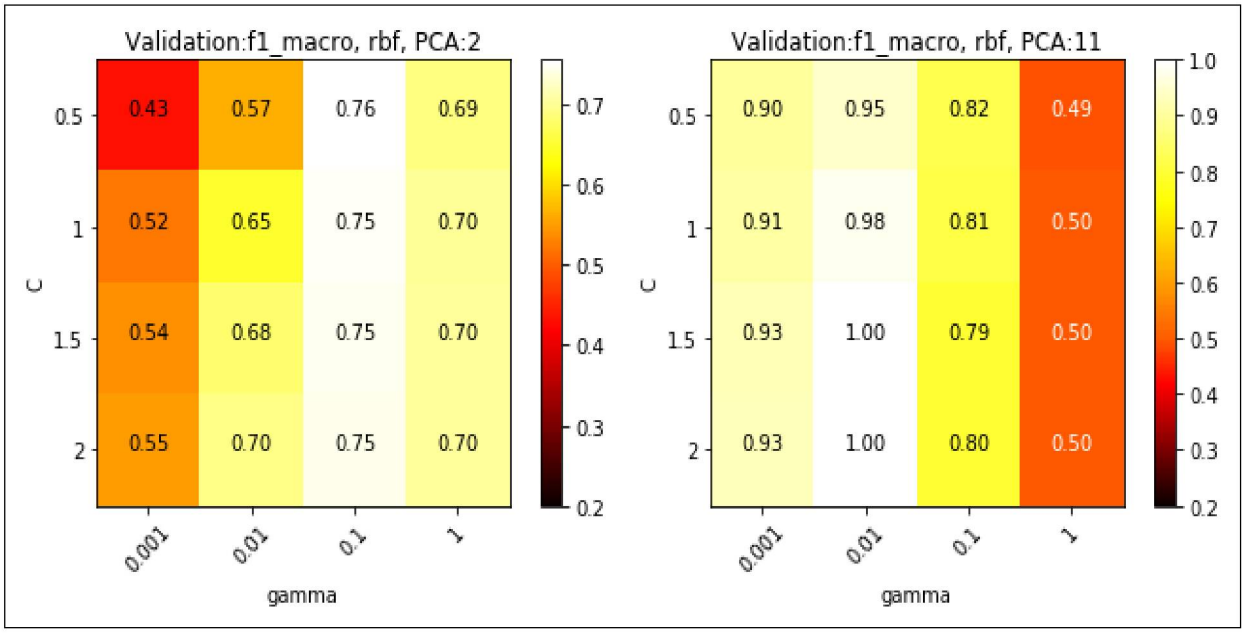
The graphs show the model's mean test score for the given evaluation metric. *Left*: model scores using 2 principal components, *Right*: model scores using 11 principal components.



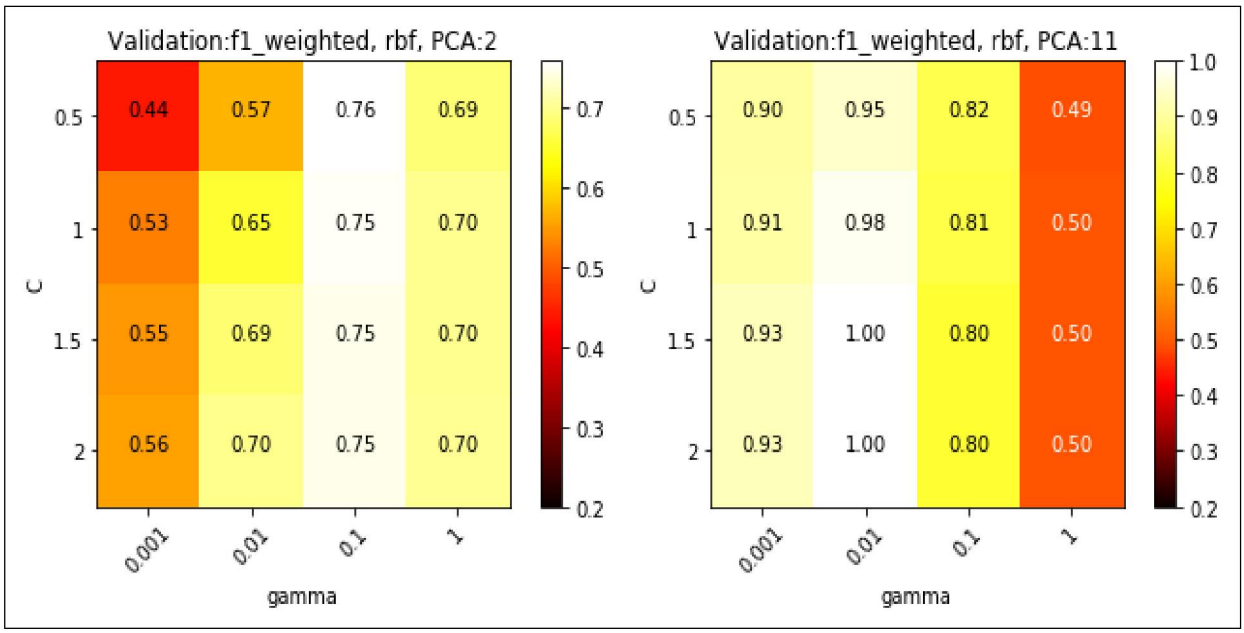
From Fig 8, (a) *precision_ macro*



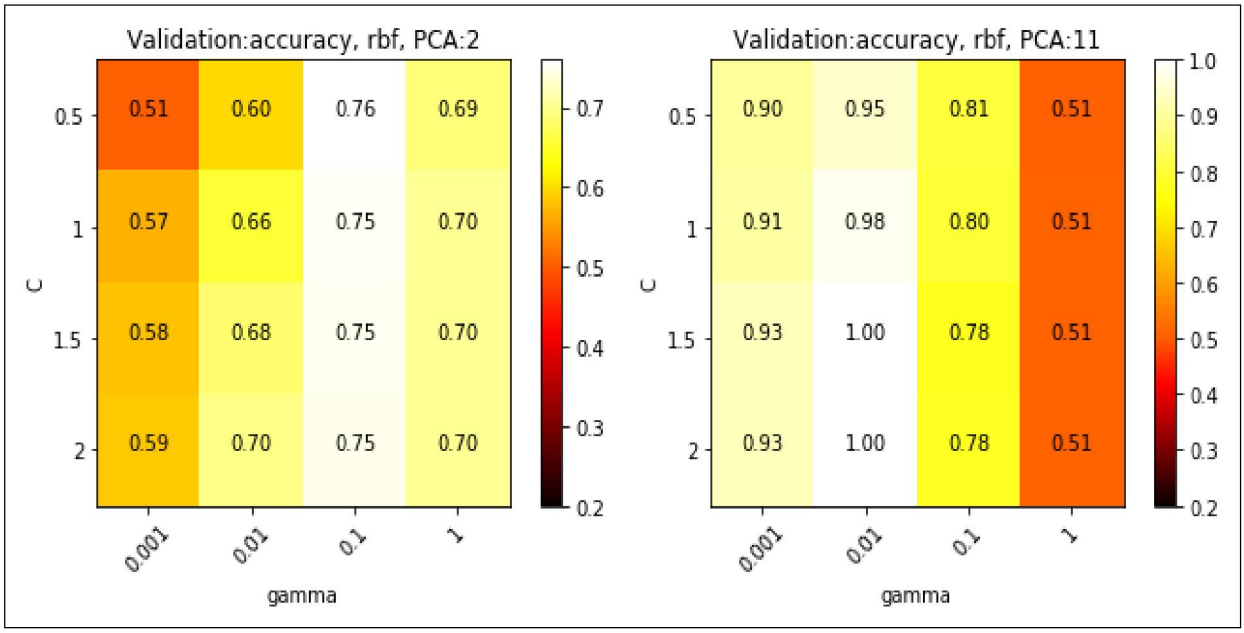
From Fig 8, (b) *recall_ macro*



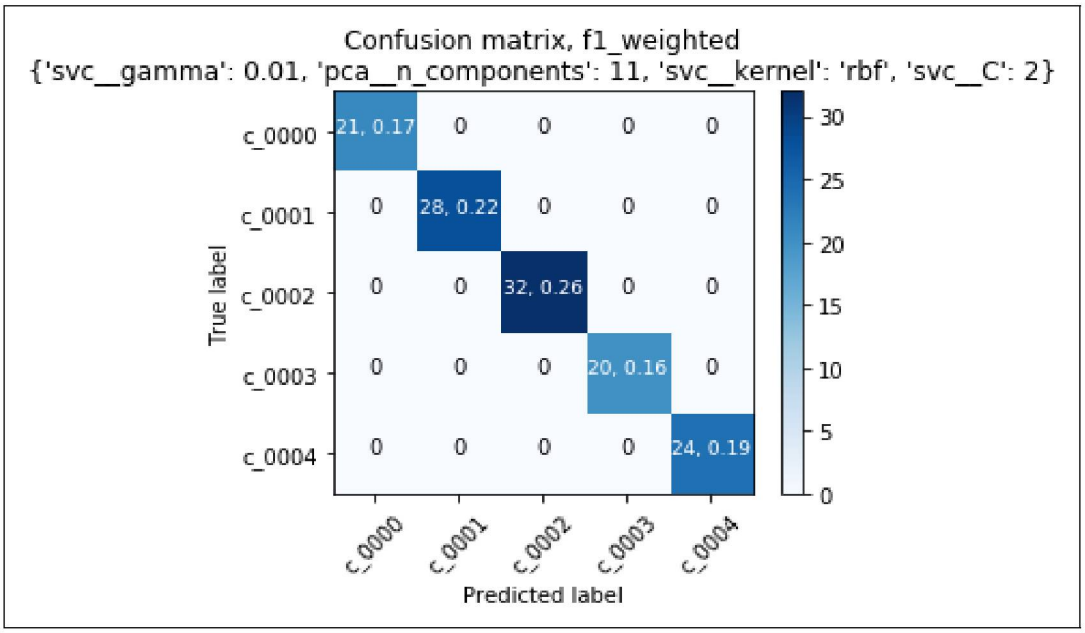
From Fig 8, (c) *f1_ macro*



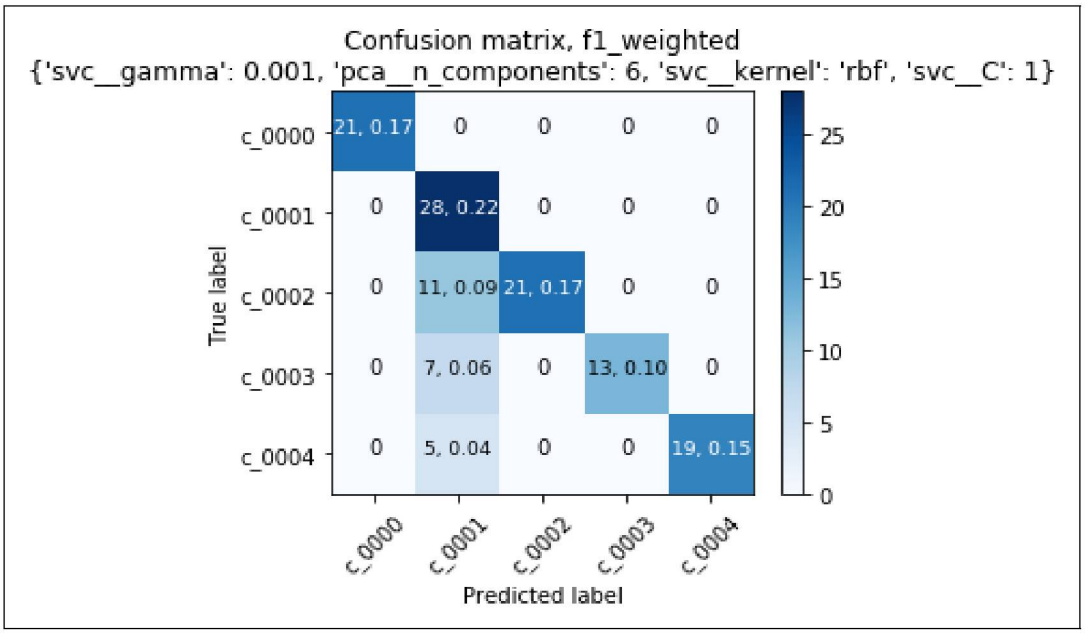
From Fig 8, (d) *f1_ weighted*



From Fig 8, (e) *accuracy*



From Fig 8, (f) classification best model evaluated with *f1_ weighted* metric



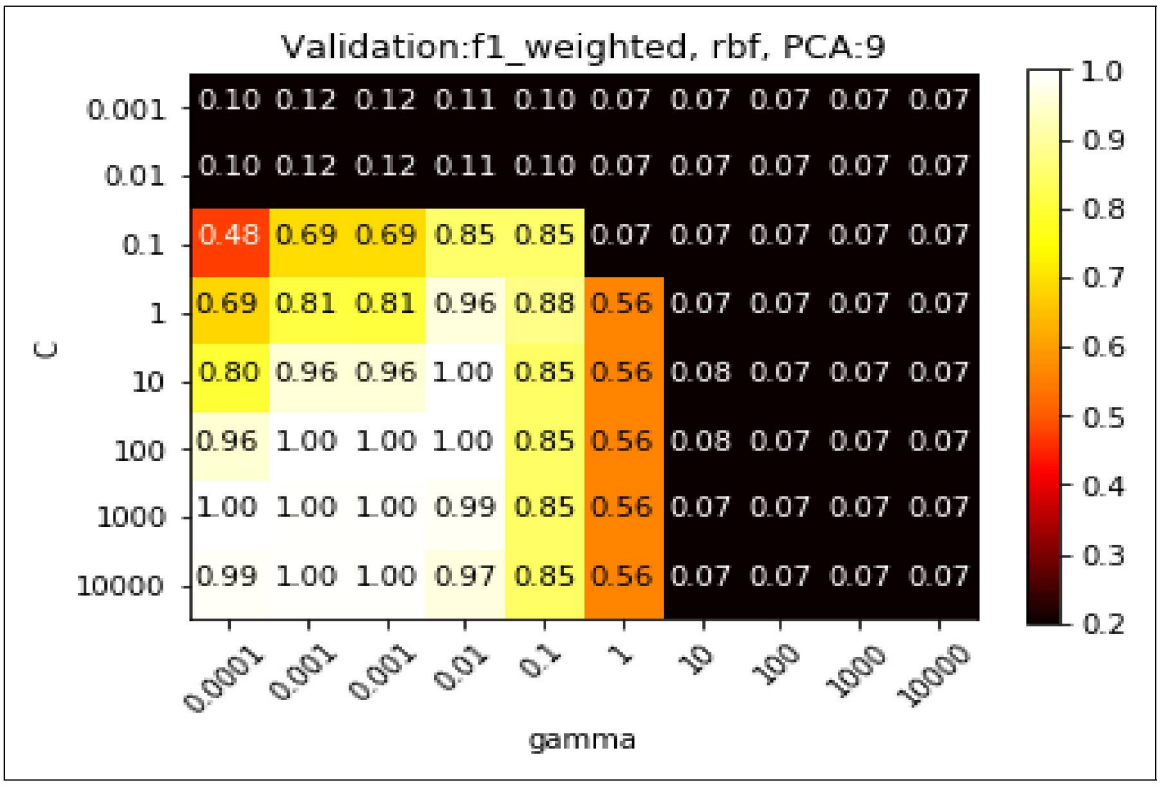
From Fig 8, (g) Results from a poorly performing classifier

	precision	recall	f1-score	support
0	1.00	1.00	1.00	21
1	0.55	1.00	0.71	28
2	1.00	0.66	0.79	32
3	1.00	0.65	0.79	20
4	1.00	0.79	0.88	24
avg / total	0.90	0.82	0.83	125

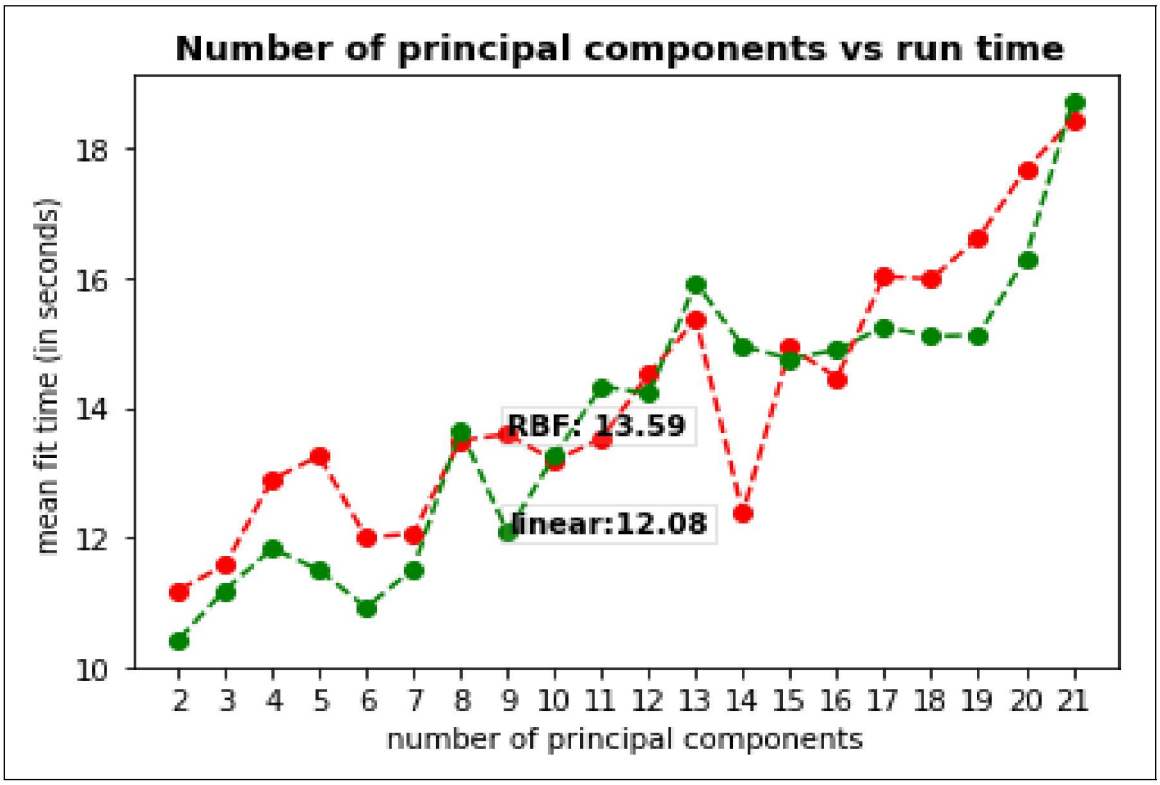
From Fig 8, (h) Metric scores of the poorly performing classifier in Figure C(g)

Appendix D Experiment 3 - Hyperparameter optimization

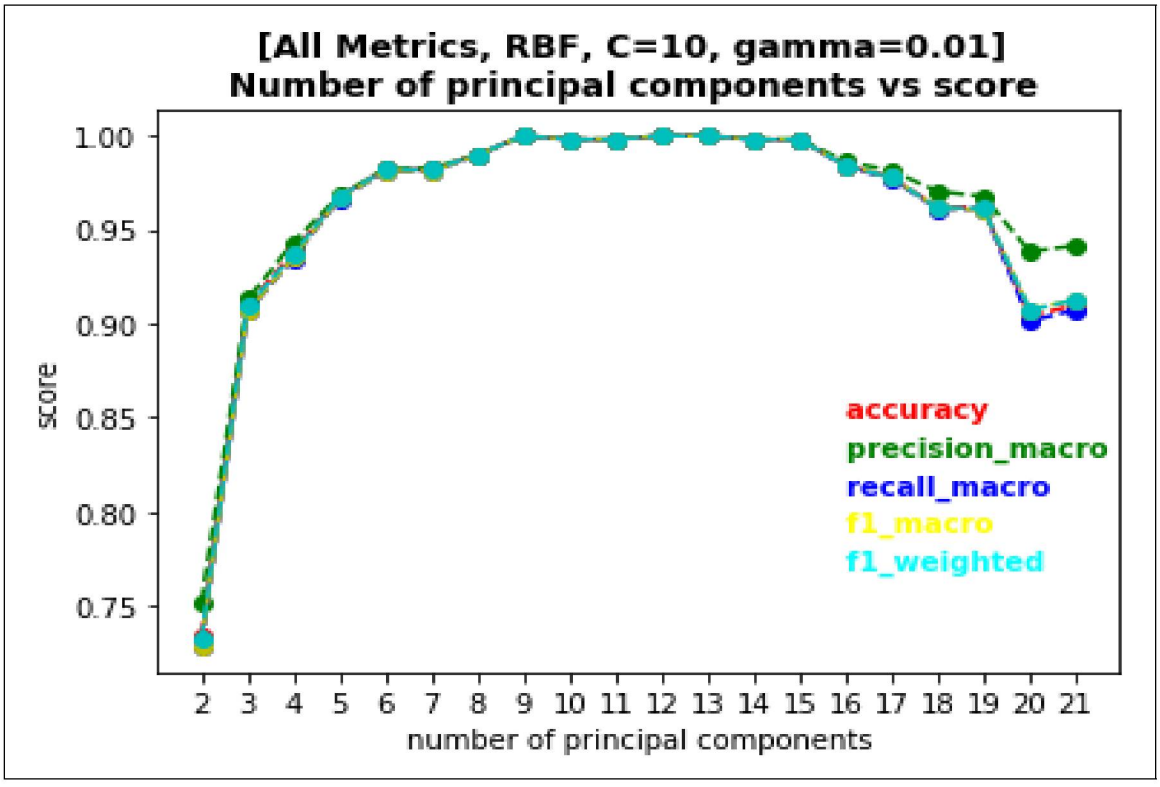
The figures show the different facets of the model's performance based on different hyperparameters.



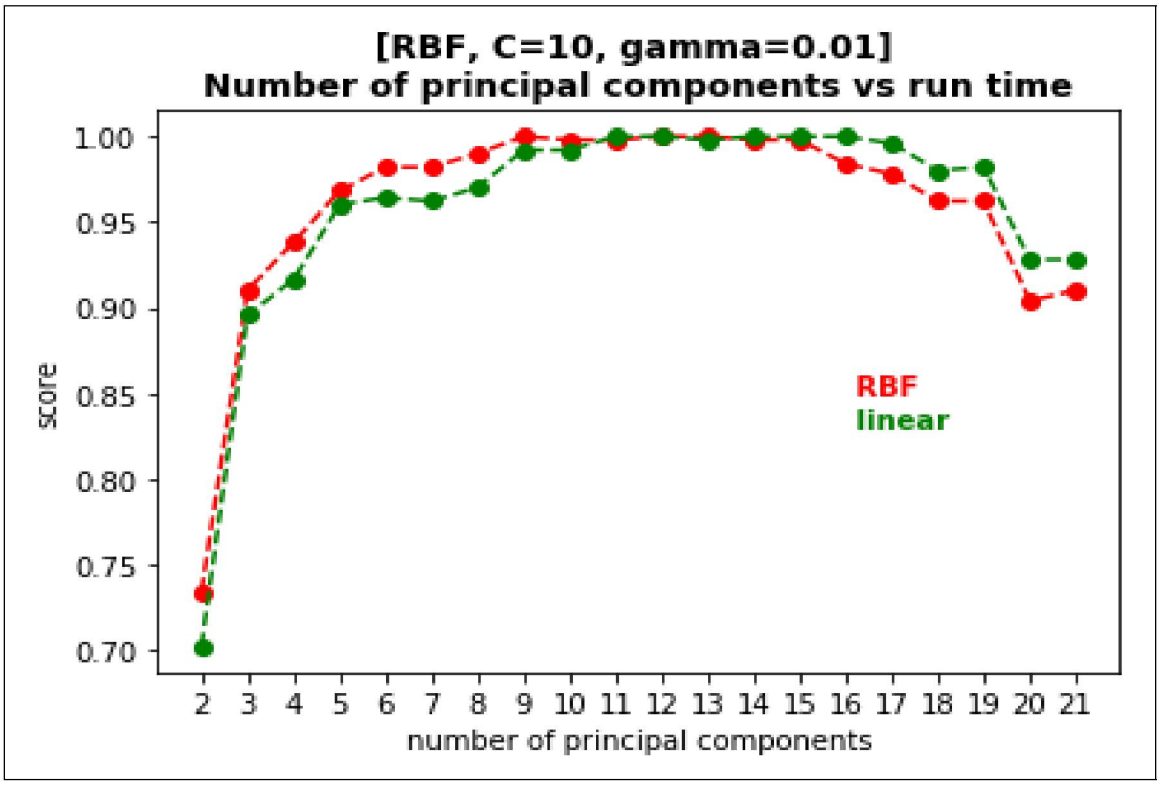
From Fig 9, (a) *f1_weighted* scores



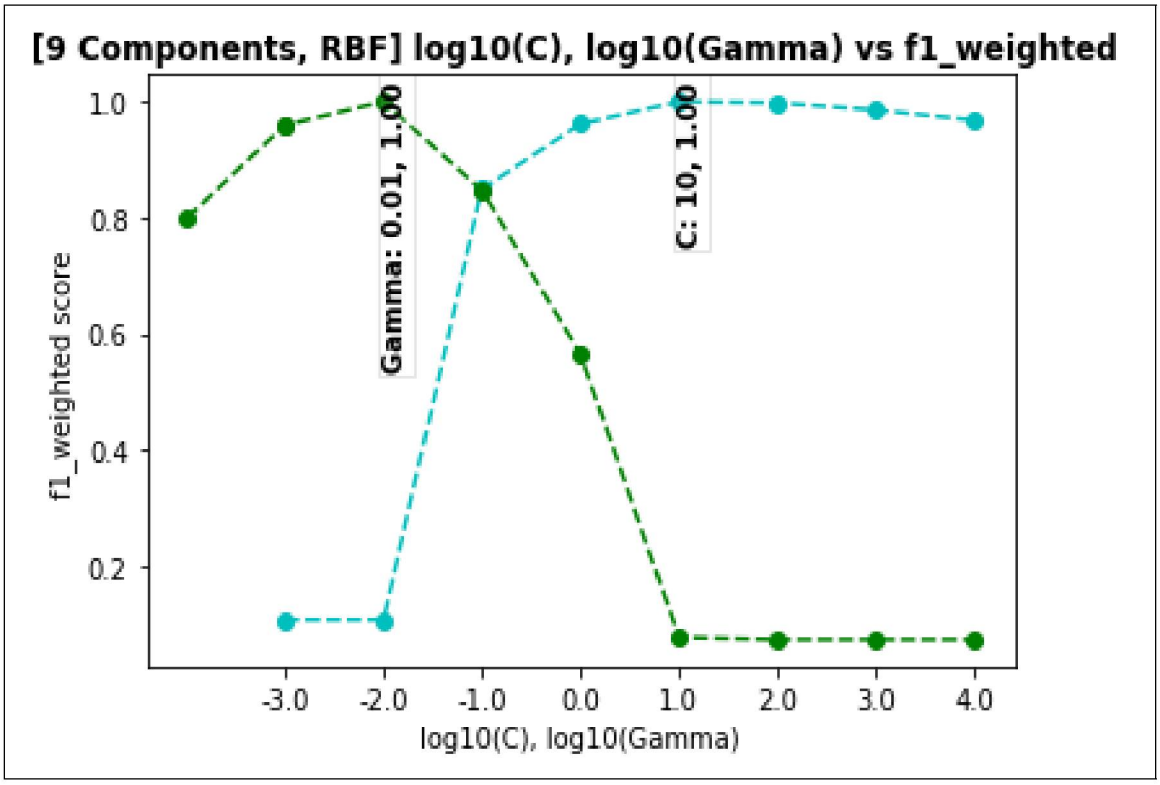
From Fig 9, (b) Run time results



From Fig 9, (c) Scoring for model with RBF kernel, $C = 10$, and $\gamma = 0.01$ for varying number of principal components



From Fig 9, (d) *kernel function comparisons (rbf, linear)*



From Fig 9, (e) C and γ parameters

References

[1] Wikipedia contributors, "Precision and recall — Wikipedia, the free encyclopedia," 2018, (2018-05-24). [Online]. Available: https://en.wikipedia.org/w/index.php?title=Precision_and_recall&oldid=835396495

[2] Moran, Janeth, Michelle Moeller, "Data Mining: Classification Techniques," California State University Channel Islands, Computer Science Program Technical Report CS-CI-TR-01-2013, 2013.

[3] About us. (2015-04-20). [Online]. Available: https://www.csuci.edu/sri/about.htm

[4] Research. (2015-04-20). [Online]. Available: https://www.csuci.edu/sri/research/

[5] K. Scrivnor, "CI Rainbow: A Flexible WSN for Environmental Data," December 2016. [Online]. Available: http://hdl.handle.net/10211.3/184081

[6] Wikipedia contributors, "Supervised learning — Wikipedia, the free encyclopedia," 2017, (2017-11-09). [Online]. Available: https://en.wikipedia.org/w/index.php?title=Supervised_learning&oldid=841771601

[7] ——, "Unsupervised learning — Wikipedia, the free encyclopedia," 2017, (2017-11-27). [Online]. Available: https://en.wikipedia.org/w/index.php?title=Unsupervised_learning&oldid=841565016

[8] ——, "Wav — Wikipedia, the free encyclopedia," 2018, (2015-10-04). [Online]. Available: https://en.wikipedia.org/w/index.php?title=WAV&oldid=838323456

[9] "Lecture 11.4 — Machine Learning System Design | Trading Off Precision And Recall — [Andrew Ng]," 2016. [Online]. Available: https://www.youtube.com/watch?v=W5meQnGACGo

[10] Cowling, Michael, Renate Sitte, "Comparison of techniques for environment sound recognition," in *Pattern Recognition Letters*, vol. 24, no. 15, Nov 2003, pp. 2895–2907.

[11] M. A. Acevedo, C. J. Corrada-Bravo, H. Corrada-Bravo, L. J. Villanueva-Rivera, and T. M. Aide, "Automated classification of bird and amphibian calls using machine learning: A comparison of methods," *Ecological Informatics*, vol. 4, no. 4, pp. 206–214, 2009.

[12] Pascale Giraudet and Olivier Dufour and Thierry Artieres and Hervé Glotin, "Clusterized Mel Filter Cepstral Coefficients and Support Vector Machines for Bird Song Identification," in *Soundscape Semiotics*, H. Glotin, Ed. Rijeka: InTech, 2014, ch. 5. [Online]. Available: https://doi.org/10.5772/56872

[13] "FindSounds: Search the Web for Sounds," (2015-01-02). [Online]. Available: http://www.findsounds.com/types.html

[14] "SoX - Sound eXchange | HomePage," Feb 2015, (2015-06-13). [Online]. Available: http://sox.sourceforge.net

[15] Cournapeau, David, "SciKits," (2015-06-15). [Online]. Available: https://scikits.appspot.com/audiolab

[16] "YAAFE, an Easy to Use and Efficient Audio Feature Extraction Software," B.Mathieu, S.Essid, T.Fillon, J.Prado, G.Richard, proceedings of the 11th ISMIR conference, Utrecht, Netherlands, 2010.

[17] Blondel, Mathieu, et al., "scikit-learn: Machine Learning in Python," (2015-06-15). [Online]. Available: http://scikit-learn.org/stable/

[18] "Discovery of Sound in the Sea," The University of Rhode Island, 2018, (2018-03-24). [Online]. Available: https://dosits.org/science/measurement/what-sounds-can-we-hear

[19] Dr. K. Forinash, "Sound: An Interactive eBook on the Physics of Sound, 10G Animal Hearing," (2018-04-16). [Online]. Available: https://soundphysics.ius.edu/?page_id=2657

[20] "Mel Frequency Cepstral Coefficient (MFCC) tutorial. Practical Cryptography," (2015-02-02). [Online]. Available: http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/#references

[21] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.