



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

**CLASSIFICATION OF SURFACE VESSELS
USING UNDERWATER ACOUSTIC
DATA AND MACHINE LEARNING**

by

John M. Henderson

September 2023

Thesis Advisor:
Co-Advisor:

Monique P. Fargues
Kay L. Gemba

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE September 2023	3. REPORT TYPE AND DATES COVERED Master's thesis	
4. TITLE AND SUBTITLE CLASSIFICATION OF SURFACE VESSELS USING UNDERWATER ACOUSTIC DATA AND MACHINE LEARNING			5. FUNDING NUMBERS	
6. AUTHOR(S) John M. Henderson				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			12b. DISTRIBUTION CODE A	
13. ABSTRACT (maximum 200 words) Automatic vessel classification is a highly relevant research topic, particularly for the U.S. Navy. In this study, we consider three machine learning techniques to classify maritime vessels based on their underwater noise: Gaussian mixture models, random forest, and k-nearest neighbors. The ShipsEar database, developed by Santos-Domínguez et al., was used to conduct the study. Mel-frequency cepstrum coefficients were selected for class feature characteristics to compare with previous findings presented by Santos-Domínguez et al. in their publication titled "ShipsEar: An Underwater Vessel Noise database" published in the <i>Applied Acoustics</i> journal, volume 113. Results indicate that all three methods offer a feasible solution to the classification problem. Notably, Gaussian mixture models show significant performance improvements over results achieved by Santos-Domínguez et al.				
14. SUBJECT TERMS machine learning, AI, SONAR, ShipsEar, ML, GMM			15. NUMBER OF PAGES 79	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

**CLASSIFICATION OF SURFACE VESSELS USING UNDERWATER
ACOUSTIC DATA AND MACHINE LEARNING**

John M. Henderson
Lieutenant, United States Navy
BS, Texas A&M University, College Station, 2016

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

**NAVAL POSTGRADUATE SCHOOL
September 2023**

Approved by: Monique P. Fargues
Advisor

Kay L. Gemba
Co-Advisor

Douglas J. Fouts
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Automatic vessel classification is a highly relevant research topic, particularly for the U.S. Navy. In this study, we consider three machine learning techniques to classify maritime vessels based on their underwater noise: Gaussian mixture models, random forest, and k-nearest neighbors. The ShipsEar database, developed by Santos-Domínguez et al., was used to conduct the study. Mel-frequency cepstrum coefficients were selected for class feature characteristics to compare with previous findings presented by Santos-Domínguez et al. in their publication titled “ShipsEar: An Underwater Vessel Noise Database” published in the *Applied Acoustics* journal, volume 113. Results indicate that all three methods offer a feasible solution to the classification problem. Notably, Gaussian mixture models show significant performance improvements over results achieved by Santos-Domínguez et al.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

1	Introduction	1
1.1	Motivation	1
1.2	Previous Work	1
1.3	Database.	2
1.4	Methodology Overview.	3
1.5	Feature Selection	5
1.6	Chapter Overview	5
2	Machine Learning Algorithms	7
2.1	Gaussian Mixture Models	7
2.2	K-Nearest Neighbors	10
2.3	Random Forests	11
2.4	Cross-Validation.	12
3	Implementation	15
3.1	Preprocessing.	15
3.2	Mel-Frequency Cepstrum Coefficients	16
3.3	Training, Validation, & Recombination.	19
4	Results	21
4.1	Performance Metrics	21
4.2	Results	24
4.3	Number of Gaussians Per Gaussian Mixture Model	26
5	Conclusion and Recommendations	29
	Appendix A MATLAB Code	31
	Appendix B Confusion Matrices	55

List of References	59
Initial Distribution List	61

List of Figures

Figure 1.1	Screenshot of ShipsEar database interface descriptor.	3
Figure 1.2	Methodology overview diagram.	4
Figure 2.1	Expectation-maximization algorithm.	9
Figure 2.2	k-nearest neighbor example.	10
Figure 2.3	Binary decision tree.	11
Figure 2.4	Random forest algorithm.	12
Figure 2.5	k-fold cross-validation.	13
Figure 3.1	Class A: short-time Fourier transform spectrogram.	15
Figure 3.2	Class A: Periodogram power spectral density.	16
Figure 3.3	MFCC Extraction.	17
Figure 3.4	Triangular filter bank with 40 coefficients.	17
Figure 3.5	Mesh plot: Class A Mel-frequency cepstrum coefficients.	18
Figure 3.6	Random Forest Out-of-bag classification error.	19
Figure 4.1	Macro-averaging precision and recall.	23
Figure 4.2	Classification Rate vs. number of MFCC features used.	24
Figure 4.3	Macro-averaged precision vs. number of MFCC features used.	25
Figure 4.4	Macro-averaged recall vs. number of MFCC features used.	26
Figure 4.5	GMM classification rate vs. number of Gaussians per GMM.	27
Figure B.1	GMM Confusion Matrix.	55
Figure B.2	KNN Confusion Matrix.	56

Figure B.3	RF Confusion Matrix.	57
------------	------------------------------	----

List of Tables

Table 5.1	Results Summary Table.	29
-----------	--------------------------------	----

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

GMM	Gaussian mixture model
SONAR	sound navigation and ranging
AIS	Automatic Identification System
MFCC	Mel-frequency cepstrum coefficient
DFT	discrete Fourier transform
DCT	discrete cosine transform
STFT	short-time Fourier transform
PDF	probability density function
EM	expectation-maximization
MAP	maximum a posteriori
KNN	k-nearest neighbors
RF	random forest
CV	cross-validation
ML	machine learning
AI	artificial intelligence
RORO	roll-on/roll-off

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

I would like to thank Prof. Monique Fargues for her expert guidance and patient encouragement offered to me throughout this challenging project. Also, thank you to Prof. Kai Gemba for being my second reader and providing excellent feedback on my thesis. I am deeply grateful and humbled to have been granted the opportunity to further my education in the exceptionally influential field of electrical engineering.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:

Introduction

1.1 Motivation

With recent groundbreaking improvements in machine learning (ML) and artificial intelligence (AI), the U.S. Navy has a vested interest in the challenge of automated ship detection and classification. One of the most viable methods to accomplish this task is by using acoustics. Sound travels much further underwater than in air, and in the case of submarines, it is one of the only viable signals by which they may be classified. Acoustic analysis and vessel classification are critical skills that historically have been learned through many hours of training and experience. Improving automated classification technology would increase effectiveness across multiple warfare domains. The benefits of automated vessel classification are not, however, limited to military applications. Compared to the number of commercial vessels, the number of military vessels is extremely small. Maritime traffic control across the world would also benefit from improvements to automated vessel detection and classification, especially in major ports and choke points where traffic is congested and traffic-controlling assets are limited. Our goal in this study is to investigate the viability of using ML to develop automatic ship classifiers based on underwater acoustics, evaluate their effectiveness, and compare them to previous work reported on the database selected for our study.

1.2 Previous Work

Over the past couple of decades there has been significant research devoted to classification of acoustic signals using ML. Broadly speaking, this research may be divided into two different approaches based on feature selection. The first approach, commonly referred to as the *traditional* approach in the literature, is characterized by using hand-crafted features for classification. Examples of hand-crafted features include frequency-domain spectra used by Liu et al. [1] and cepstral coefficients used to train Gaussian mixture models (GMM) by Das et al. [2] and Santos-Domínguez et al. [3]. The second approach is characterized by using neural networks to train models on vast amounts of unlabeled data and then extracting fea-

tures using a small amount of labeled data. This is useful because of the scarcity of labeled datasets. The feasibility of using neural networks as underwater acoustic classifiers has been demonstrated by Lou et al. [4] and Ke et al. [5]. By selecting mel-frequency cepstral coefficients as features, our research presented in this paper takes the traditional approach for feature selection and classification.

1.3 Database

The database used for our study was collected by Santos-Domínguez et al. [3] and is referred to as “ShipsEar: An Underwater Vessel Noise Database.” The database contains 90 recordings that were taken in 2012 and 2013 off the coast of northwest Spain, near the port of Vigo. The recordings were sampled using three identical hydrophones that contained A/D sigma-delta converters with 24-bit bit-depth and a sampling frequency of 52 734 Hz. The recording sessions were documented in a comprehensive log, which included relevant information including, but not limited to, type of vessel, GPS data, AIS (Automatic Identification System) data, and weather conditions. The recordings consisted of a variety of vessels ranging from small motorboats and sailboats to ocean liners and merchant ships.

Among the 90 recordings, 12 were taken in the absence of vessel noise so that environmental characteristics could be extracted. Figure 1.1 shows a screenshot of the ShipsEar database web-based interface descriptor.

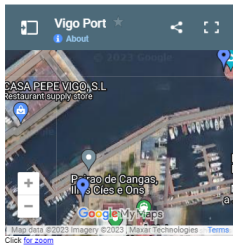
Id	Name	Type	Pic	Localization	Date	H_G_D	Video	Chanel depth	Wind	AIS link	Distance
80	Adricristuy	Dredger			2012-10-04 14:00:00	(3),(16),(5)	No	20	Not available	AIS data	350 m
10	Mar de Onza (Leaving)	Passengers			2013-07-10 00:00:00	(2),(4),(2.85)	NO	4.8	0	AIS data	<50

Figure 1.1. Screenshot of ShipsEar database interface descriptor. Source: [6]

1.4 Methodology Overview

Santos-Domínguez et al. [3] created the ShipsEar database and applied Gaussian mixture model (GMM) classifiers to the data collected to investigate the feasibility of automating the classification task [3]. Mel-frequency cepstrum coefficients (MFCC) generated from the audio recordings were selected as features. In this study, we first used the ShipsEar database and created our own GMM classifier to validate Santos-Domínguez et al. findings. Next, we considered two other ML techniques and compared resulting performances.

The overall classifier design process, including training and testing, is demonstrated in Figure 1.2. First, we removed the signal bias for each audio signal. Next, each audio signal was split into short data segments and MFCC coefficients computed on each segment of data separately. In our study, the signal partitioning step and the Fourier transform step present in the generation of the MFCC coefficients were implemented by using the short-time Fourier transform operation. For implementing the STFT operation, we used a one second Hamming window with 50% overlap between windows to ensure no loss of information. Each data segment was individually labeled by class and specific file number. Note this labeling

stage was essential to allow for individual classification decisions obtained for separate data segments to be recombined in order for the classifiers to provide decisions for complete audio signals. Then the features (MFCCs) were extracted from each data segment. Next, we created the feature vector by combining the MFCCs for each audio file into a matrix. In the following stage, feature sets were split into training and testing sets, which were used to train and test each ML algorithm. After training and validation, individual data predictions obtained on each separate data segments were recombined to provide the overall classification decision for each complete individual audio signal present in the database.

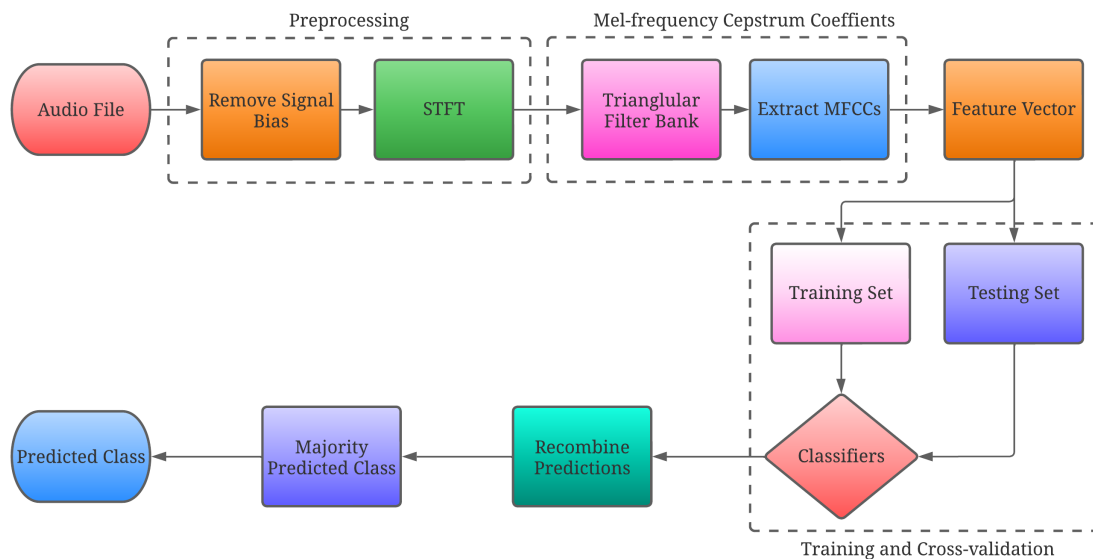


Figure 1.2. Methodology overview diagram.

In this study we used the same type of grouping as that used in [3], which partitioned the audio files into five classes: A, B, C, D, and E. Classes A, B, and D are a super-set of classes of ships. The groupings were determined by relative ship size, which expects the sounds profiles of similarly sized ships to be similar. Class E contains environmental noise only.

- Class A: fishing boats, trawlers, mussel boats, tugboats and dredgers
- Class B: motorboats, pilot boats and sailboats
- Class C: passenger ferries
- Class D: ocean liners and roll-on/roll-off (RORO) vessels
- Class E: background noise recordings

1.5 Feature Selection

MFCCs are a commonly used features in speech recognition and audio processing and were used in our study to validate the results of Santos-Domínguez et al.

MFCCs capture the spectral envelope of the signal, which is useful in representing the characteristics of the signal in a compact form. They are derived from a signal by transforming the signal from the time domain to the frequency domain using the discrete Fourier transform (DFT), then mapping the frequency spectrum to the mel scale, which is a perceptually-based frequency scale that models the human auditory system. Note in our case, we used a custom frequency scale based on analysis of the frequency content of audio from vessels. Finally, the spectrum was rectified and transformed using the discrete cosine transform (DCT) to extract the cepstral coefficients.

1.6 Chapter Overview

In Chapter 2, we present the theory behind each ML algorithm used. In Chapter 3, we discuss the implementation of MFCCs and feature extraction. In Chapter 4, we report results for each algorithm considered in the study and compare their performances. And finally in Chapter 5, we present our conclusions and recommendations for related future research.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:

Machine Learning Algorithms

In this chapter, we provide an introduction to the theory and background of the three ML methods used in our study: GMMs, k-nearest neighbors (KNN), and random forests (RF). We also discuss our method of cross-validation: k -folds cross-validation.

2.1 Gaussian Mixture Models

Gaussian mixture models model a data set by partitioning it into multiple Gaussian distributions called components [7]. As shown by Nabney, GMMs contain some given number of components whose statistical parameters (mean and covariance) are initialized to random values. The GMM is then fitted to a data set using the Expectation-maximization (EM) algorithm, an iterative method developed by Dempster et al. designed to find local maximum a posteriori (MAP) approximations of the statistical parameters required to fit each Gaussian component to the data set [8].

For M components in d dimensions, the probability density function (PDF) of a GMM is defined by Nabney [7] as

$$p(\mathbf{x}) = \sum_{j=1}^M P(j)p(\mathbf{x}|j), \quad (2.1)$$

where the mixing coefficients, $P(j)$, represent the proportion of data for a given individual component, $\mathbf{x}$ is an observation vector of length d , and $p(\mathbf{x}|j)$ represents the j th individual component PDF. A specific component density function is defined as

$$p(\mathbf{x}|j) = \frac{1}{(2\pi^{d/2})|\Sigma_j|^{1/2}} \exp \left\{ -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_j)\Sigma_j^{-1}(\mathbf{x} - \boldsymbol{\mu}_j)^T \right\}, \quad (2.2)$$

where Σ_j is the covariance matrix of dimension $d \times d$ and $\boldsymbol{\mu}_j$ is the mean vector for each component PDF [7].

2.1.1 GMM Classifier Example

Of the three ML schemes considered in this study, GMMs are conceptually the most difficult to understand. As a result, we included a simple example to illustrate the basic idea behind GMMs. In this example, we consider a two-dimensional data set that contains two easily distinguishable groupings of data (yellow and red data scatters) shown in Figure 2.1. A GMM with two components in two dimensions is fitted to the data using the EM algorithm. As described by Dempster et al. [8], the EM algorithm is an iterative approach that alternates between two steps: the E-step (expectation) and the M-step (maximization). First, the statistical parameters of the component PDFs (mean and covariance) are initialized using the k-means++ algorithm [9]. K-means++ is an algorithm used to find the initial values, also called *seeds*, for the statistical parameter vectors of the GMM, the mean and covariance for each component distribution. These parameters define the location and structure of the centroid for each distribution.

In the E-step, for each observation, the algorithm computes the posterior probabilities of component memberships. That is, in this specific example, it calculates the probabilities of an observation belonging to each of the two components.

In the M-step, the algorithm updates the mean and covariance of the component model based on the probabilities calculated in the E-step. The update is performed using a process called maximum likelihood estimation, which maximizes the likelihood of observing the observed data using the probabilities calculated in the E-step.

The algorithm continues to alternate between the E-step and M-step until convergence, meaning that the model parameters have reached a stable state and no further significant improvement is observed. The resulting estimates of the model parameters are then used to make predictions or to perform further analysis. Figure 2.1 contains an animated demonstration of the EM algorithm for a GMM with two components in two dimensions.

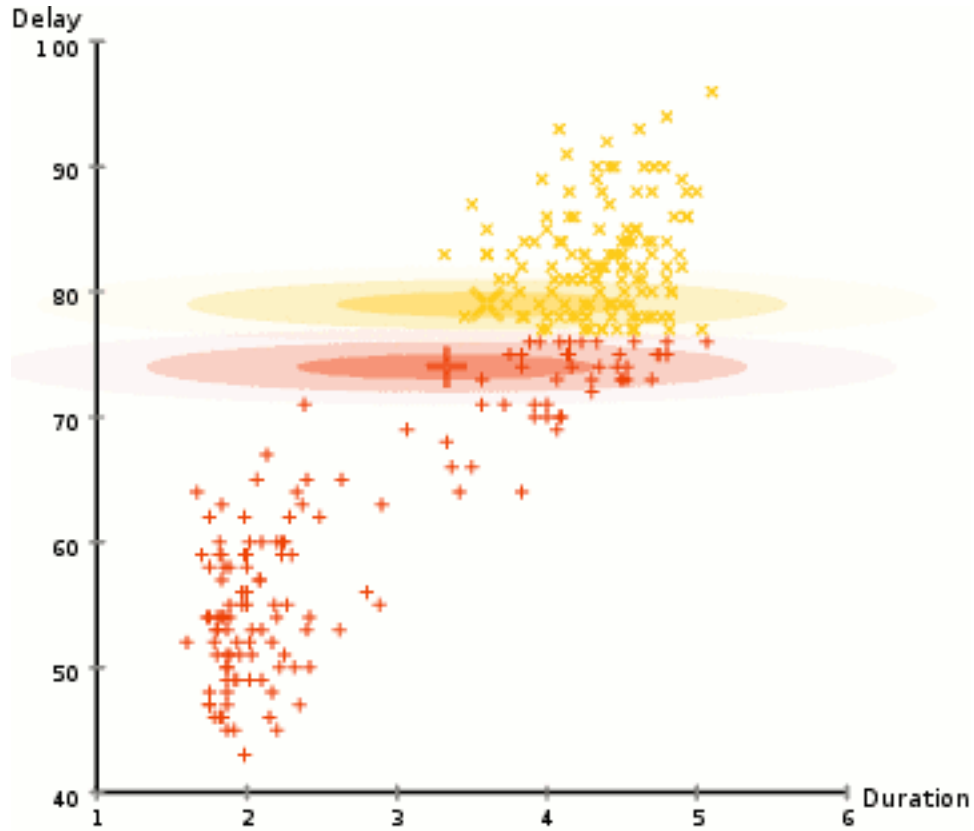


Figure 2.1. Fitting two GMMs using the EM algorithm. Source: [10]

Once the GMM has been fitted to the data set, the PDF for the entire model is fully determined by combining the individual PDFs of each component. At this point, the trained GMM may be used to classify an input data set as belonging to one of its two distributions by using the composite PDF

$$p(\mathbf{x}) = \sum_{j=1}^2 P(j)p(\mathbf{x}|j). \quad (2.3)$$

2.2 K-Nearest Neighbors

K-nearest neighbors (KNN) is a type of supervised learning algorithm where the algorithm attempts to classify new data points based on their proximity to surrounding training data points. KNN was originally developed by Evelyn Fix and Joseph Hodges in 1951 and then later expounded upon by Thomas Cover in 1967 [11].

The basic idea behind KNN is that similar things are expected to be close to each other. Hence, the algorithm tries to find the k nearest data points from the training set to the new data point and classifies it based on the majority class among these k neighbors. There are many different metrics by which the distance between neighboring points may be calculated. Euclidean distance is typically the most commonly used. Figure 2.2 provides a graphical demonstration of how the KNN algorithm works. In this example, $k = 5$. Therefore, the five nearest points are circled to indicate that they are being used to classify the new data point. Since there are three data points in Category A, and two data points in Category B, the new data point will be classified as Category A.

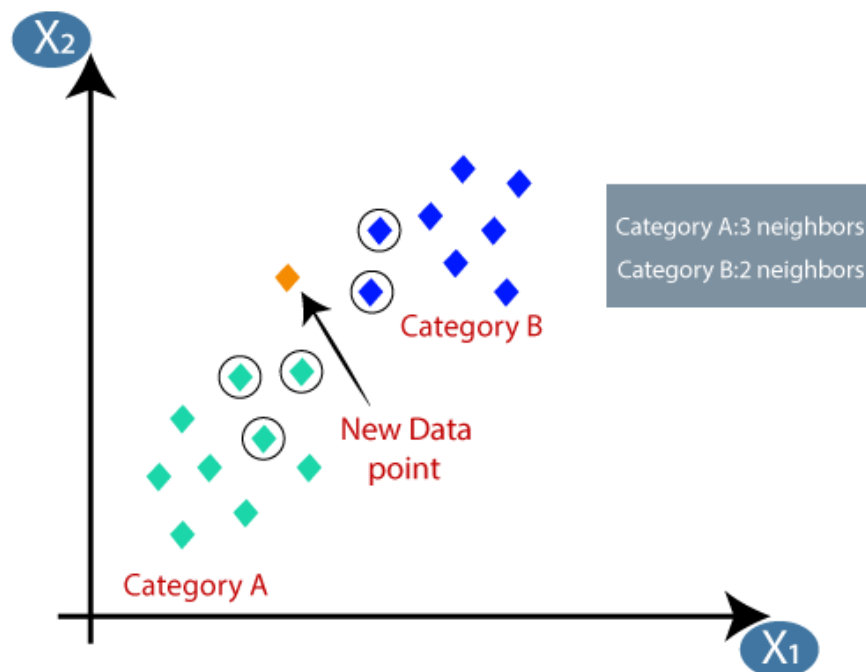


Figure 2.2. KNN example for $k = 5$. Source: [12]

2.3 Random Forests

As described by Breiman, the random forest (RF) is an ensemble ML algorithm that combines the results of multiple decision tree models to improve the accuracy and robustness of the predictions [13]. The random forest model uses multiple decision trees as its baseline model with *bagging*. Bagging comes from the combination of two words: **bootstrapping** and **aggregating**. Bootstrapping means to create multiple subsets of the original data by randomly selecting observations with replacement. These data subsets are referred to as *bootstraps*. Sampling with replacement helps ensure that each bootstrap is independent from the others, as it does not depend on previous chosen samples when sampling. Aggregating refers to taking the majority vote among all of the predictions from the decision trees in order to determine the overall prediction of the classifier. Random forests also implement another kind of bagging scheme for feature selection. For each binary decision tree, a random subset of the features is selected. This helps ensure that binary decision trees remain as uncorrelated as possible. After bootstrapping the data into subsets, a binary decision tree, shown in Figure 2.3, is constructed for each bootstrap.

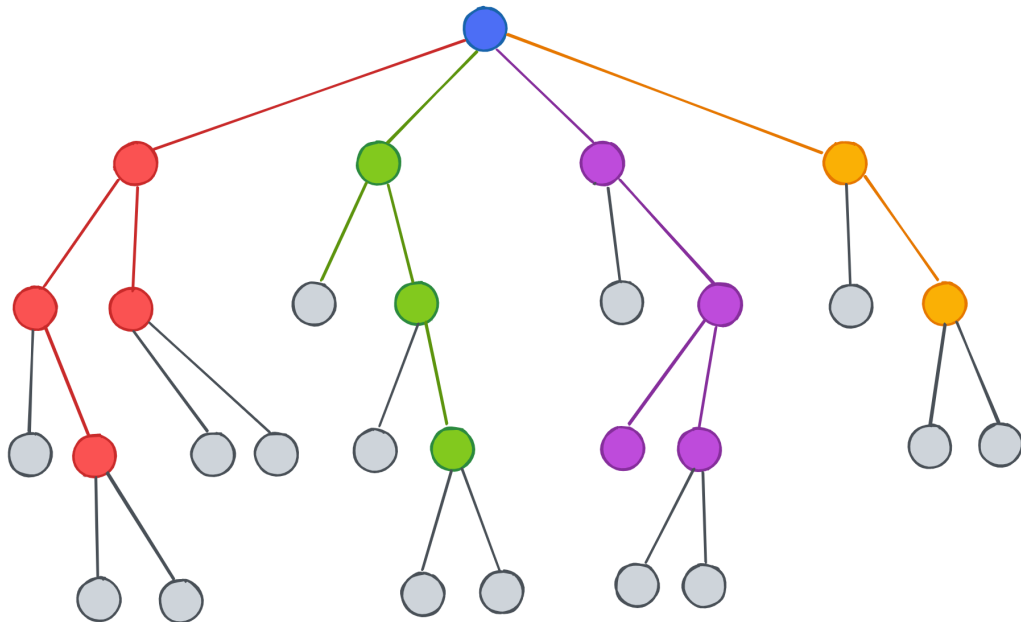


Figure 2.3. Binary decision tree. Source: [14]

A binary decision tree is created by recursively partitioning its data into subsets based on the values of the input features. Each node in the tree represents a decision based on the value of a particular feature, and each branch represents a possible outcome of that decision. The leaves of the tree represent the final decision or prediction. At each step, the decision tree algorithm selects the feature that best splits the data into subsets that are more homogeneous with respect to the target variable. The algorithm uses various metrics, such as Gini impurity or information gain, to evaluate the quality of a split. After bootstrapping, the decision trees are built using each bootstrap. Next, the decision trees are given the validation data to make predictions on. Finally, the overall decision is obtained using majority rule [13]. Figure 2.4 provides an illustration of the random forest algorithm.

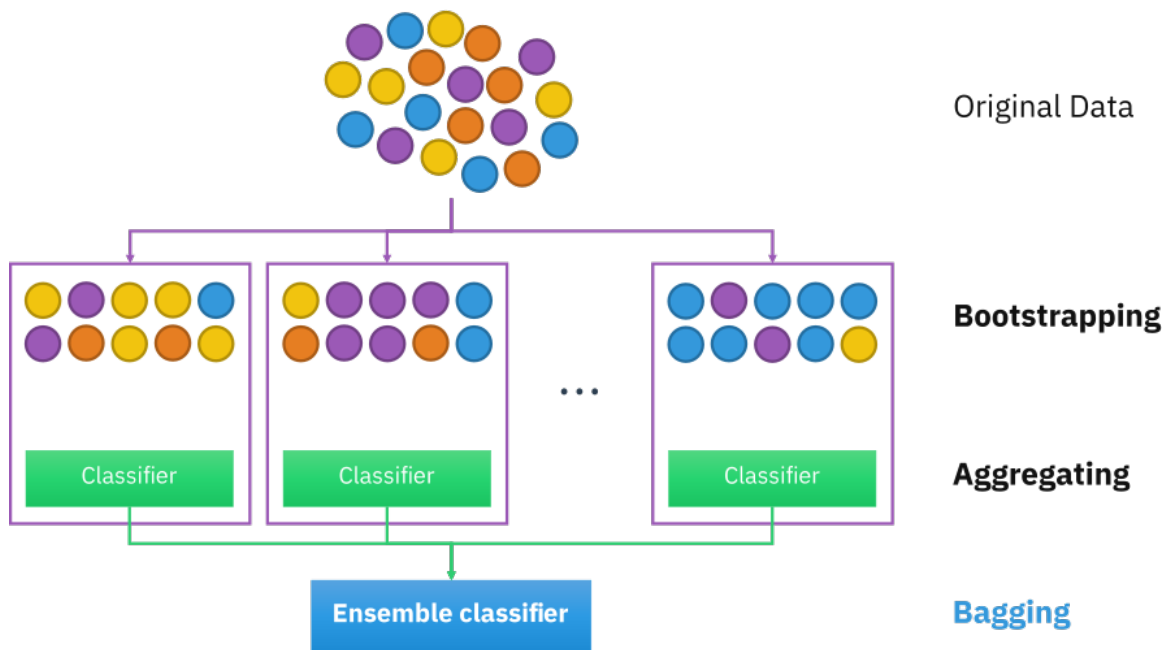


Figure 2.4. Random forest algorithm. Source: [15]

2.4 Cross-Validation

K-fold cross-validation is a useful technique to avoid over-fitting and to obtain a more accurate estimate of the performance [16], as well as to decrease the likelihood that performance is highly dependent on the specific partitioning used. K-fold cross-validation divides the data set into k equal parts, or “folds.” Each model is trained on $k - 1$ folds and validated

on the remaining fold. This process is repeated k times, each time using different folds for training and validation.

The results from each fold are then averaged to provide an estimate of the aggregate performance of the model. The choice of k can vary depending on the size of the data set and the computational resources available, but typically k is set in the range of 5 to 10. See Figure 2.5 for a graphical depiction of the k -fold cross-validation process [17].

In this chapter we provided a brief introduction to classification approaches considered in this study. In the next chapter we will discuss pre-processing and implementation of the MFCCs using MATLAB.

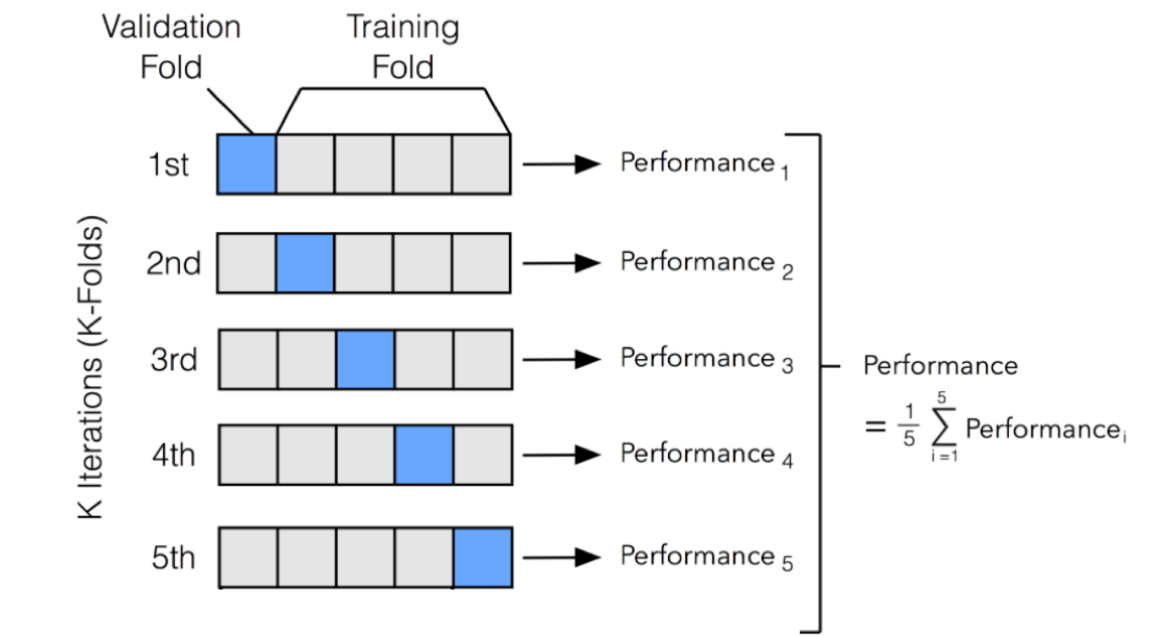


Figure 2.5. k -fold cross-validation scheme for $k = 5$. Source: [17]

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3: Implementation

In this chapter we discuss the specific implementation of feature extraction through MFCCs as well as the implementation of each ML algorithm. The ML classifiers were implemented using the MATLAB Statistics and ML Toolbox [18].

3.1 Preprocessing

Before we extracted the MFCC features from each audio file, we removed the signal bias from each audio file by subtracting its mean to account for minute bias differences between hydrophones. Then, we took the STFT of each audio file to split it up into segments and to convert it from the time-domain to the frequency-domain. Figures 3.1 and 3.2 demonstrate the average behavior of Class A via the spectrogram and periodogram, respectively. We used a one second Hamming window with 50% overlap. We experimented with several different overlap lengths, which will be discussed further in Chapter 4.

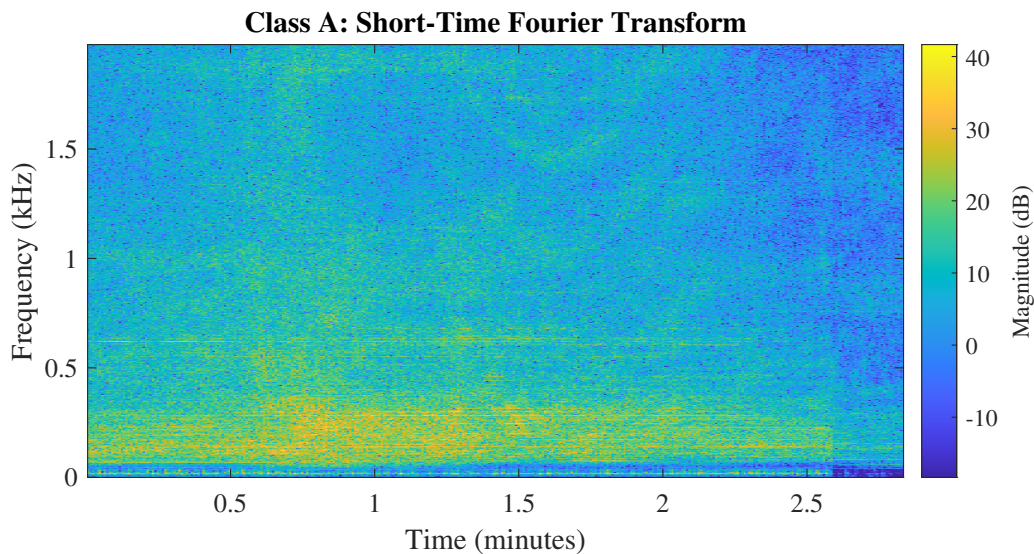


Figure 3.1. Class A: short-time Fourier transform spectrogram.

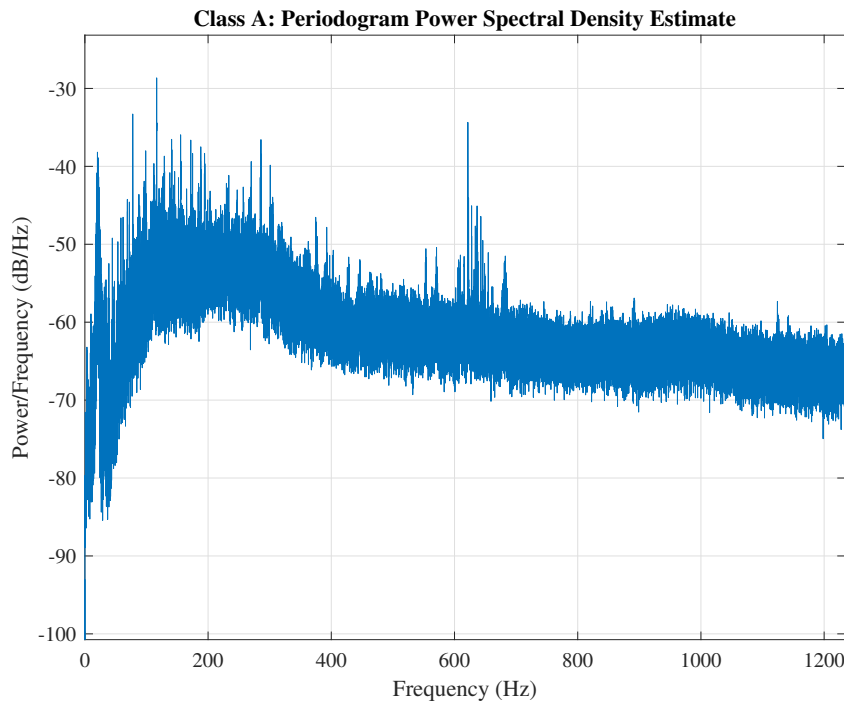


Figure 3.2. Class A: Periodogram power spectral density.

Notice how most of the energy in the signal is located below 1 kHz. This is expected due to the characteristics of maritime engines, as well as the rapid attenuation of higher frequency components. Additionally, although Figure 3.2 only covers up to 1200 Hz, the overall trend is the same; all subsequent higher frequencies have decreasing magnitude.

3.2 Mel-Frequency Cepstrum Coefficients

Next, we extracted the MFCCs from each audio signal. This computation was accomplished in the following three steps: taking the STFT, applying a triangular filter bank, and then using the MATLAB function `cepstralCoefficients`. These steps are visually demonstrated in Figure 3.3.

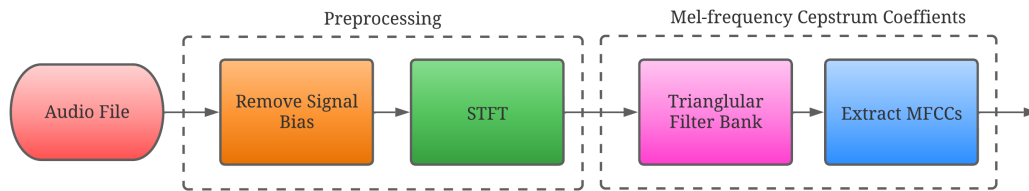


Figure 3.3. MFCC Extraction.

After applying the STFT to each audio file using a one second Hamming window with 50% overlap, we applied a triangular filter bank to the spectrum in order to map the frequency components to the mel spectrum. See Figure 3.4 for an illustration of the triangular filter bank frequency response obtained using 40 coefficients in the range of 0 to 26 367 Hz, half of the sampling frequency.

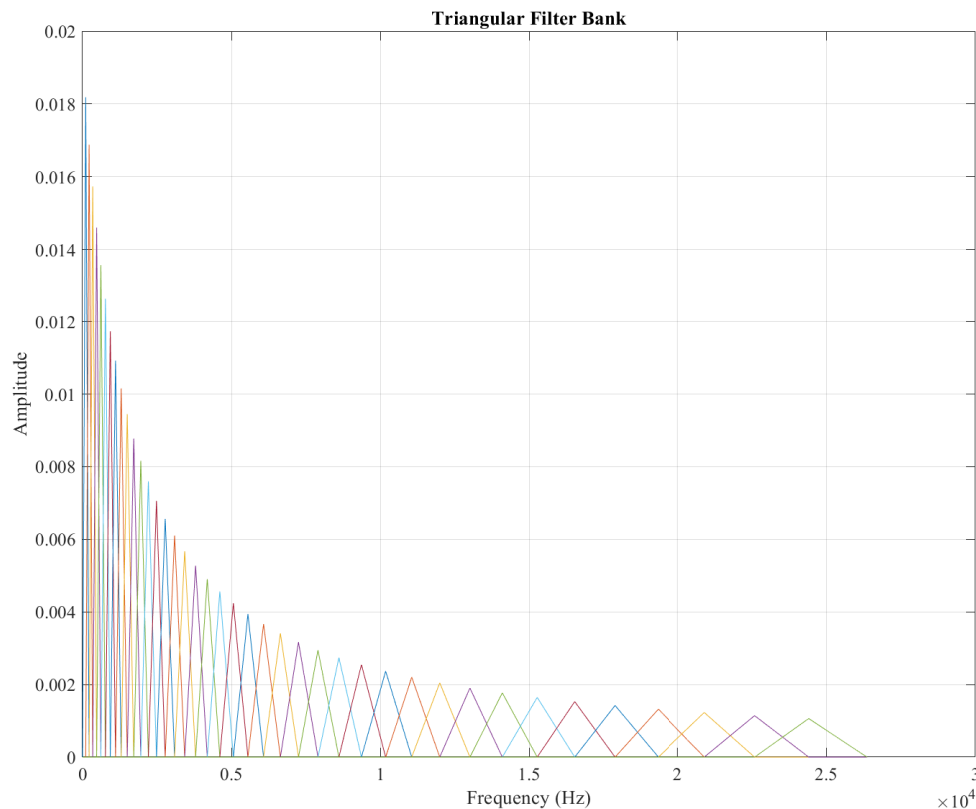


Figure 3.4. Triangular filter bank with 40 coefficients.

After extracting the MFCCs for all 90 audio files, we created the feature vector by concatenating the MFCCs for each audio file into a matrix of dimension $N \times d$, where N is the number of observations and d is the number of dimensions (number of MFCCs). We created a mesh plot on a class basis to observe the overall trend of the MFCCs. The mesh plot for Class A is shown below in Figure 3.5. Mesh plots for the other four classes displayed similar overall trends.

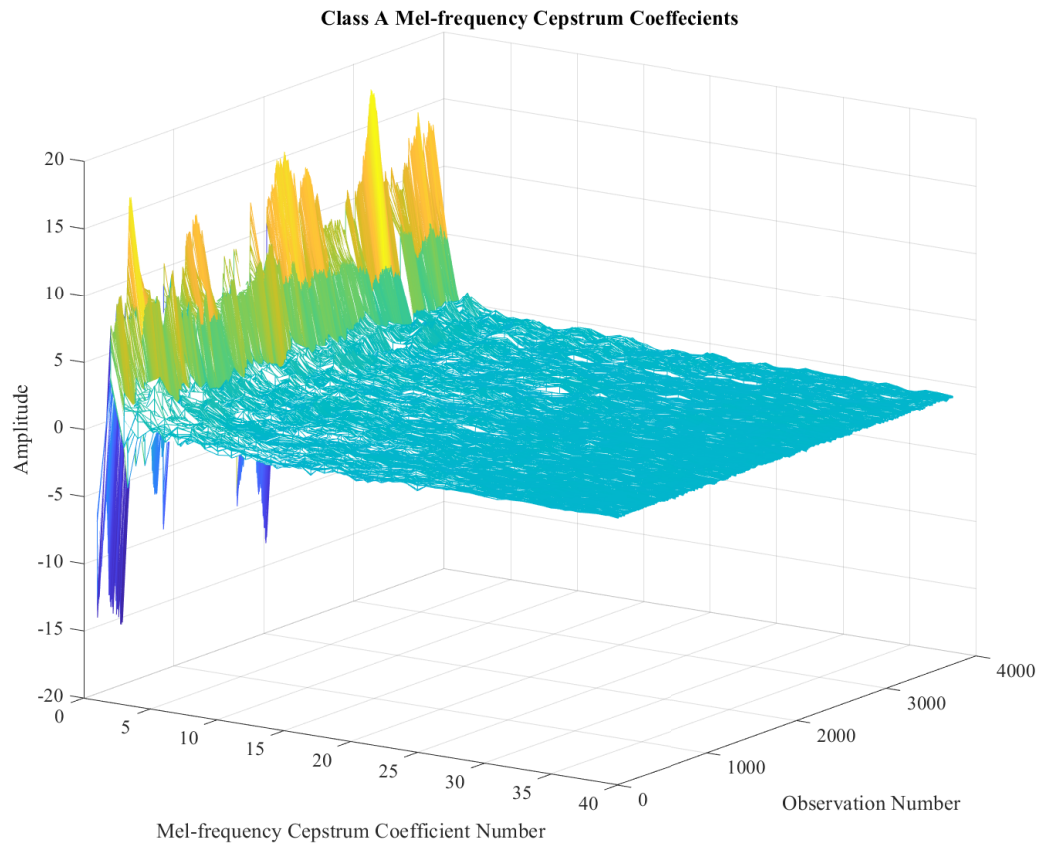


Figure 3.5. Mesh plot: Class A Mel-frequency cepstrum coefficients.

The number of triangular filters used to extract the MFCCs directly corresponds to the number of MFCCs extracted. Notice how the amplitude of the coefficients tends to rapidly decrease as the coefficient number increases. Again, this demonstrates that the majority of the energy in the audio signals is contained in the lower frequency range (<1 kHz).

3.3 Training, Validation, & Recombination

Once feature vectors (MFCCs) were built, we partitioned our data to create mutually exclusive training and testing sets. This step was accomplished by using the MATLAB function `cvpartition`, which creates five iterations of the data set to be used in $k = 5$ fold cross-validation, exactly as shown in Figure 2.5. Then, we ran each algorithm discussed five times, once for each fold.

3.3.1 Random Forest

We used the MATLAB function `TreeBagger` with 500 trees for the RF classifier implementation. We analyzed the out-of-bag classification error to determine when the algorithm had stabilized. Figure 3.6 shows that the RF algorithm achieved stability at around 150 trees.

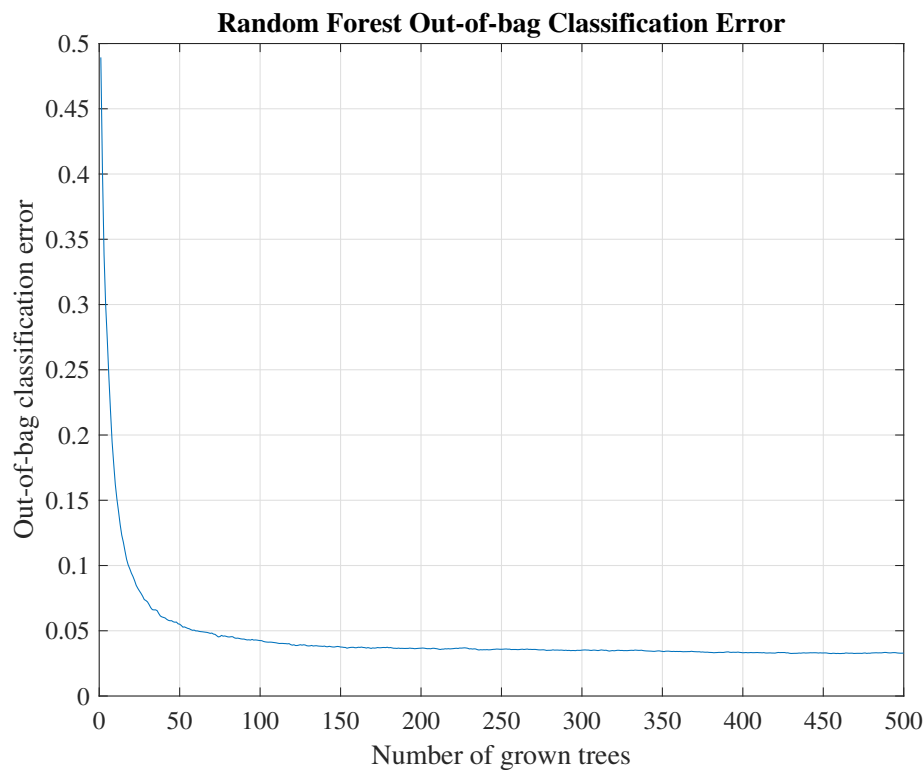


Figure 3.6. Random Forest Out-of-bag classification error.

3.3.2 K-Nearest Neighbors

KNN was implemented using the MATLAB function `fitcknn` using $k = 10$ neighbors to implement a *medium* KNN classifier.

3.3.3 Gaussian Mixture Model

GMM was implemented using MATLAB function `fitgmdist` using one GMM for each class, and six Gaussians per GMM.

3.3.4 Recombination

Recall that each classifier makes predictions on individual short data segments, as explained earlier in this chapter, and overall classification decisions are desired at the audio file level. Thus, first, decisions obtained for individual data segments corresponding to the same parent audio file are grouped together. The final prediction decision obtained for each audio file is determined by applying the majority rule to each respective groups of data segment decisions.

In this chapter we discussed the implementation of feature extraction and cross-validation in MATLAB. The complete MATLAB code can be found in Appendix A. In the next chapter we will present and analyze performance results obtained for each classifier.

CHAPTER 4:

Results

In this chapter we present the results of the three algorithms used: Gaussian mixture models, the random forest algorithm, and the k-nearest neighbors algorithm. We will discuss three main performance metrics: precision, recall, and classification rate using k-fold cross-validation for $k = 5$. We compare classification performances and evaluate their dependencies to the changes in sampling frequency, the number of MFCC features, and the number of Gaussian components used in the GMM classifier.

4.1 Performance Metrics

In this section, we discuss the three performance metrics applied to evaluate the overall classifier performance; classification rate, precision, and recall.

4.1.1 Classification Rate

The classification rate is the usual default performance measure used to evaluate classifier performance. It is simply the ratio of the number of correct results to the total number of observations. For binary classification, where the two classes are referred to as positives and negatives, the classification rate is defined as

$$\text{Classification Rate} = \frac{TP + TN}{TO}, \quad (4.1)$$

where TP is the number of true positives, TN is the number of true negatives, and TO is the number of total observations.

However, classification rate alone is not well suited for multi-class classification problems where the number of observations between classes is not evenly distributed. Therefore, we also consider the metrics *precision* and *recall* first defined for the binary classification scenario.

4.1.2 Precision

Precision, also known as positive prediction rate, is the fraction of true positive results to the total number of results predicted. It is a useful metric for determining the model accuracy in predicting *positive* results.

$$Precision = \frac{TP}{TP + FP}, \quad (4.2)$$

where TP is the number of true positives, and FP is the number of false positives.

Precision was initially defined for binary classification problems and later extended to multi-class problems by using *macro-averaging*. The macro-averaged precision is derived by calculating the precision on a per-class basis and then averaging the results to get the overall classifier precision.

4.1.3 Recall

Recall, also known as sensitivity, is the fraction of the number of true positive results to the total number of relevant results. Recall is a useful metric for determining what proportion of true positives were correctly identified. For binary classification problems, recall is defined as

$$Recall = \frac{TP}{TP + FN}, \quad (4.3)$$

where TP is the number of true positives, and FN is the number of false negatives.

Similar to precision, the definition of recall can be extended to the multi-class scenario to obtain the macro-averaged recall quantity by calculating the recall on a per-class basis and then averaging the results to get the overall classifier recall. Figure 4.1 provides a step-by-step example of how to calculate macro-averaged precision and recall using data from our multi-class classification problem. After the individual calculations were performed for precision and recall, the macro-averaged results were calculated using equations 4.4–4.5.

GMM Classifier Confusion Matrix						
True Class	A	13	4			$P_A \quad \frac{13}{13+4}$
	B		16	3		$P_B \quad \frac{16}{16+3}$
	C			29	1	$P_C \quad \frac{29}{29+1}$
	D	1			11	$P_D \quad \frac{11}{1+11}$
	E					$P_E \quad \frac{12}{12}$
Predicted Class						
	A	B	C	D	E	
	R_A	R_B	R_C	R_D	R_E	
	$\frac{13}{13+1}$	$\frac{16}{16}$	$\frac{29}{4+3+29}$	$\frac{11}{1+11}$	$\frac{12}{12}$	

Figure 4.1. Macro-averaging precision and recall, 5-class scenario.

$$R_{GMM} = \frac{1}{5} \sum_{i=1}^5 R_i = 93\% \quad (4.4)$$

$$P_{GMM} = \frac{1}{5} \sum_{i=1}^5 P_i = 90\% \quad (4.5)$$

Additional confusion matrices for all three classifiers have been included in Appendix B.

4.2 Results

In this section, we evaluate how the number of MFCC features used impacted the resulting classifier performance. This investigation will help us understand whether classification performance is related to the frequency content of the data. Recall that the original sampling frequency of the audio files was 52 734 Hz and that a mel-frequency cepstrum coefficient number is related to the particular frequency content of that coefficient. That is, the 40 coefficients, derived from a signal sampled at 52 734 Hz, cover the frequency range 0 to 26 367 Hz. The specific frequency content for every five MFCCs is annotated on the results graphs.

In these graphs (Figures 4.2–4.4), we have ten different data points per algorithm. Each data point represents a different number of MFCCs used in the classifier. The data points are as follows: 1-5, 10, 15, 20, 30, 40. The error bars show the standard deviation for the $k = 5$ folds cross-validation scenario. The frequency annotations on the graph indicate the highest frequency the band of features covers. That is, the first five features cover up to 768 Hz, the first ten features cover up to 1721 Hz, etc.

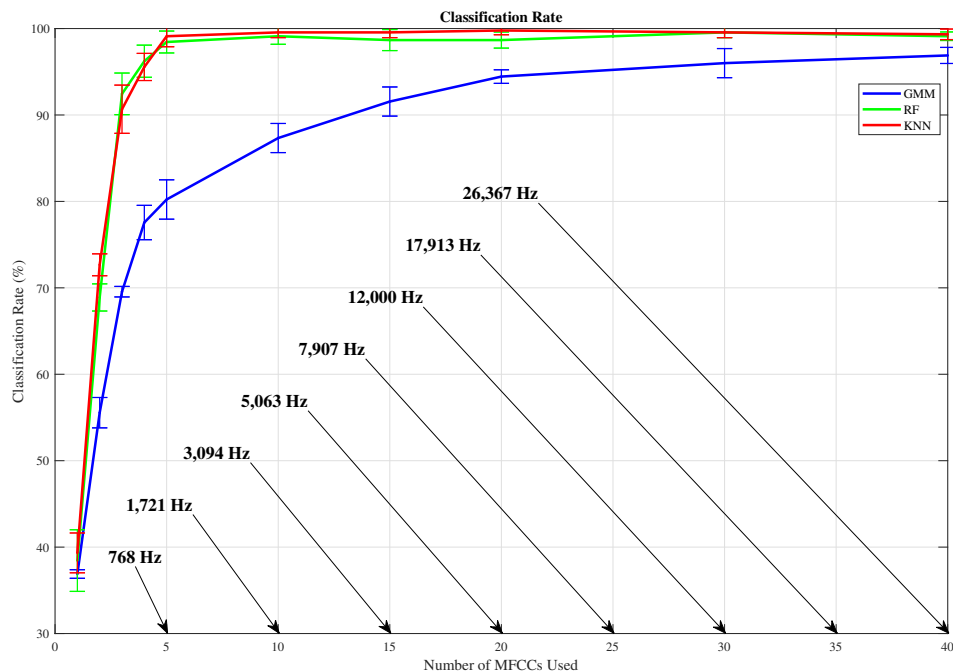


Figure 4.2. Classification Rate vs. number of MFCC features used.

Results indicate that the largest performance gains occur when using the first five MFCCs, corresponding to a maximum frequency of ~1 kHz. This finding is consistent with our expectations as well as with the results obtained by Santos-Domínguez et al. who noted that the frequency range for noise generated by ship engines is primarily confined to frequencies below 1 kHz [3]. Additionally, it is noteworthy that further increasing the number of features used results in increasingly diminishing returns for performance. However, for the GMM classifier, the classification rate gains do not fall off as quickly as with the KNN and RF models. Specifically, the GMM classifier shows significant improvement in classification rate when utilizing the first ten features as compared to the first five, whereas RF and KNN do not display significant improvement in classification rate after the first five features.

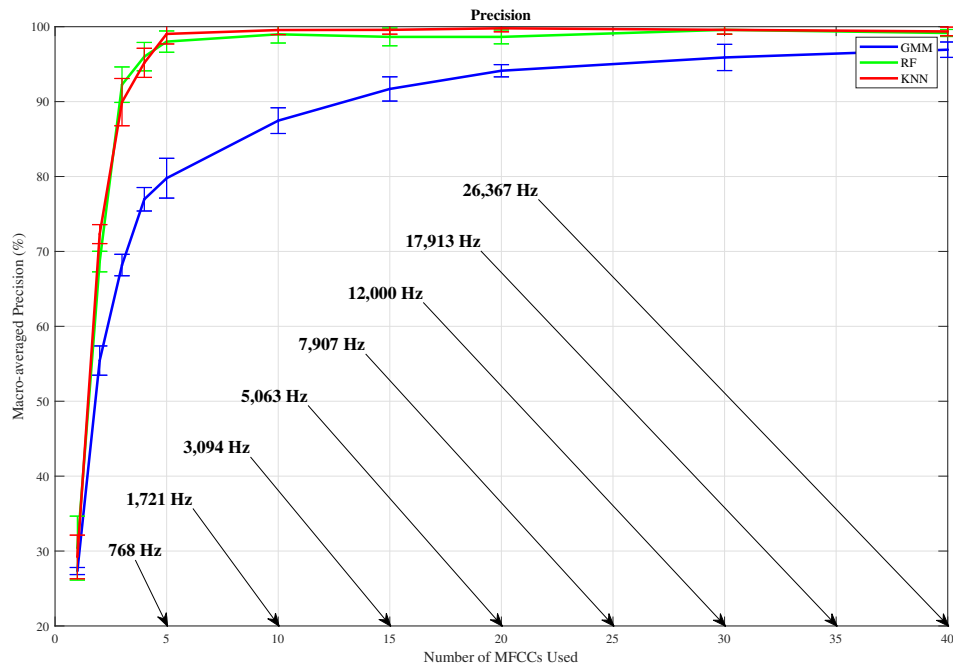


Figure 4.3. Macro-averaged precision vs. number of MFCC features used.

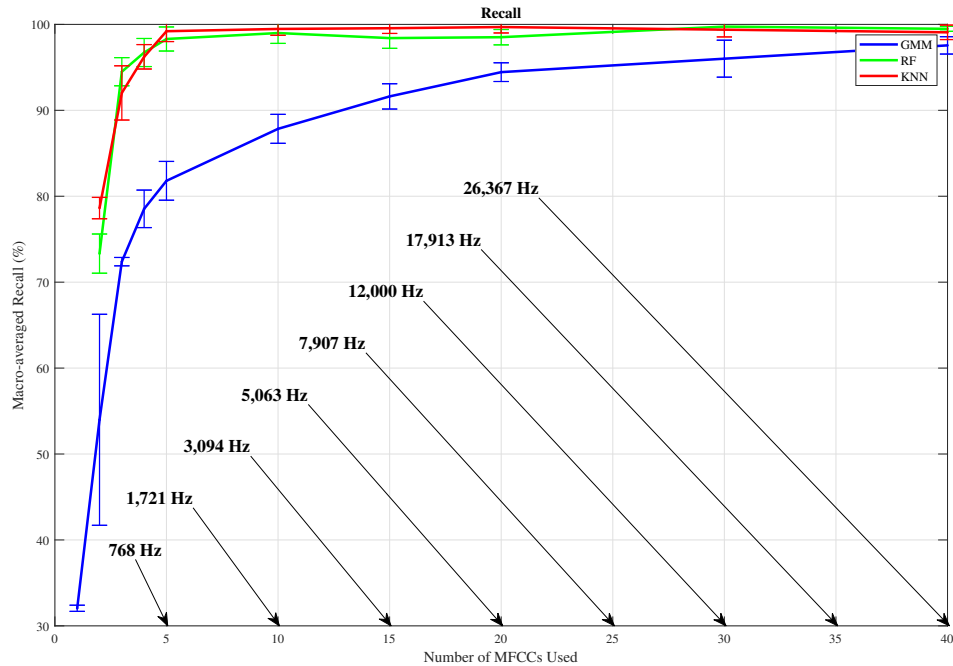


Figure 4.4. Macro-averaged recall vs. number of MFCC features used.

Precision and recall estimates follow a similar trend until around 1 kHz. Notice how all three metrics for each classifier have lower performance at low sampling frequencies with the performance gains dropping off rapidly after the 5th coefficient is used. Additionally, we also considered using the 1st and 2nd derivatives of the MFCCs, referred to as *delta* and *delta-delta* coefficients, respectively, as additional features. However, results showed no significant changes in classification rates and were not used in final results included here.

4.3 Number of Gaussians Per Gaussian Mixture Model

Santos-Domínguez et al. [3] experimentally determined that using six Gaussians per GMM model was best in their experiments. We considered various numbers of Gaussians per GMM to evaluate how that parameter impacts the resulting GMM classifier performances and results are shown in Figure 4.5.

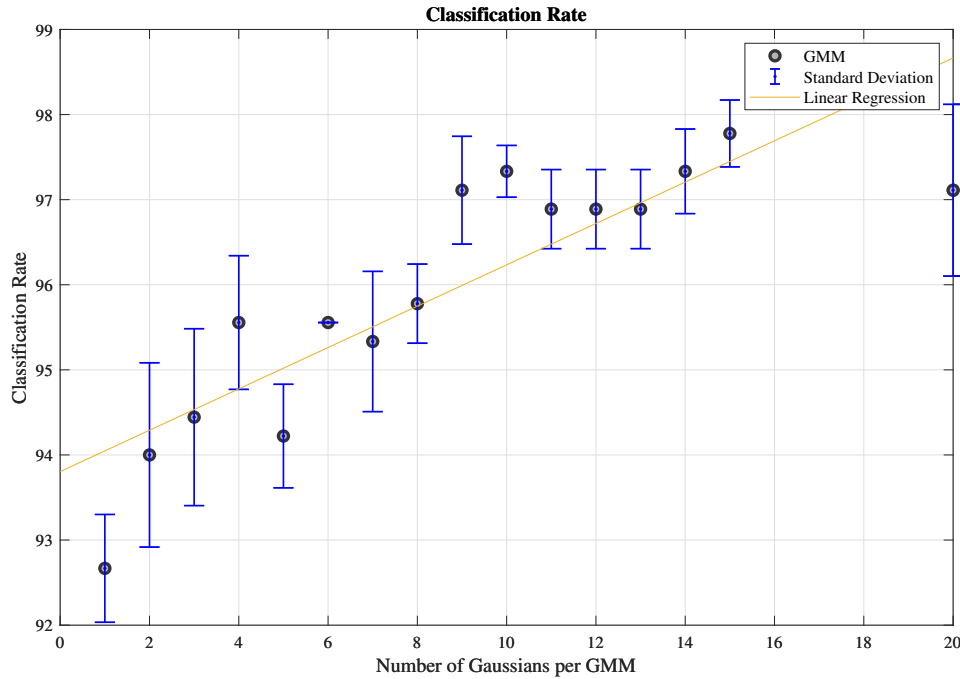


Figure 4.5. GMM classification rate vs. number of Gaussians per GMM.

Results shown in Figure 4.5 indicate a large spread in classification results. However, they also indicate some level of linear dependence between the classification rate and the number of Gaussians utilized per GMM. Interestingly, we noted the lowest variance level is obtained when using six Gaussians per GMM, which was also the number of Gaussians per GMM selected by Santos-Domínguez et al. study [3].

In this chapter we provided background information on the performance metrics considered in this study: classification rate, precision, and recall. Next, we presented the performance results obtained for each classifier considered in the study. Next, we will summarize major results and provide recommendations for further research in this area.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:

Conclusion and Recommendations

In this study, we investigated the viability of ML approaches to classify vessels from their underwater acoustic signals. In Chapter 1, we provided relevant background information on the ShipsEar database used in the study, discussed previous work conducted by the authors of that database, and briefly introduced the three ML algorithms and MFCC features applied in our study. In Chapter 2, we provided specific background information on each ML algorithm considered. In Chapter 3, we described the specific implementation utilized in our study and provided observations on the raw data and features. In Chapter 4, we described the performances metrics used to quantify classifier performances: classification rate, macro-averaged precision and recall, and provided performance results obtained during the study. Relevant results are summarized in Table 5.1.

Table 5.1. Results Summary Table.

Algorithm	Performance Metric		
	Classification Rate	Precision	Recall
GMM	96%	97%	97%
RF	98%	99%	99%
KNN	98%	99%	98%

Results indicate the KNN and RF algorithms performed the best with a 98% classification rate obtained using 40 MFCC features. However, the GMM classifier also performed well with a 96% classification rate. Thus, results indicate all three ML methods represent a potential solution to our goal of developing classifier tools from underwater recordings. Results also suggest that much of the useful information contained in the features resides in the lower frequencies (< 1kHz), as indicated by Figures 4.2–4.4, where we noted the performance gains for each algorithm rapidly drop off as the number of features is increased. This finding is consistent with our analysis of the audio spectrum as well as the MFCC behavior on a per-class basis. Recall from Figure 3.5 that the amplitude of the MFCCs rapidly

lowered as the MFCC number increased and from Figure 3.2, which shows the highest magnitude frequencies were contained below 1 kHz. This finding suggests that lower frequency features contain more useful information than higher frequency features, and is also corroborated by Santos-Domínguez et al. who noted in their study that the frequency range for noise generated by ship engines is primarily confined to frequencies below 1 kHz [3].

Our research considered five super-classes of ships created by combining multiple different types of ships into five individual classes, due to the small size of the database available for the study. Automated vessel classification is a highly active domain of research, currently somewhat constrained by the limited availability of labelled data. Increasing database sizes has the potential to produce significant contributions towards advancing capabilities in automated ship recognition.

Additionally, extensive research has been undertaken regarding the generation of synthetic data in situations where acquiring real data is challenging, as well as evaluating how synthetic data influences classifier performance. The latest advancements in deep learning, particularly through diffusion models, provide novel approaches for generating synthetic data as demonstrated by Ava Amini during her lecture at MIT [19]. Leveraging these techniques to create supplementary data and evaluating their effects on classifying ships based on underwater acoustic signatures would be highly relevant at this moment.

APPENDIX A: MATLAB Code

```
% NPS MSEE Thesis - LT John M. Henderson, June 2023
% This code will import the ShipsEar database into a struct
% with various fields
% (e.g. name, audio vector, ID, type, etc.)

clc; clear all;
path_directory='G:\My Drive\NPS\Thesis\shipsEar_AUDIOS
-20220107T162314Z-001';

% Import excel file to be used to create struct database
opts = detectImportOptions('shipsEar.xlsx');
% Set the data type for 'ID' to be type 'double' - all
% others are 'string'.
opts = setvartype(opts,'ID','double');
table = readtable('shipsEar.xlsx',opts,'ReadVariableNames',
false);
table = sortrows(table,1);
% Read all .wav files
original_files=dir([path_directory '/*.wav']);
for k = 1:90
    files(k).pathname = [path_directory '/' original_files(k)
        ].name];
    files(k).audio = audioread(files(k).pathname);
    files(k).ID = table2array(table(k,1));
    files(k).description = convertCharsToStrings(table{k,3})
        ;
    files(k).type = convertCharsToStrings(table{k,4});
% Sort the files based on ship type.
```

```

if any(files(k).type == ["Fishboat","Trawler","Mussel
    boat","Tugboat","Dredger"])
    files(k).class = 'A';
end

if any(files(k).type == ["Motorboat","Pilot ship","
    Sailboat"])
    files(k).class = 'B';
end

if any(files(k).type == "Passengers")
    files(k).class = 'C';
end

if any(files(k).type == ["Ocean liner","RORO"])
    files(k).class = 'D';
end

if any(files(k).type == "Natural ambient noise")
    files(k).class = 'E';
end
end
classA = [];
classB = [];
classC = [];
classD = [];
classE = [];
% Create class-specific structs to be used later.
for i = 1:90
    if files(i).class == 'A'
        classA = [classA,files(i)];
    end
    if files(i).class == 'B'

```

```

        classB = [classB,files(i)];
    end
    if files(i).class == 'C'
        classC = [classC,files(i)];
    end
    if files(i).class == 'D'
        classD = [classD,files(i)];
    end
    if files(i).class == 'E'
        classE = [classE,files(i)];
    end
end

%
-----

% In this next section, we begin processing the data on a
% per-audio file
% basis.

format compact
clearvars -except classA classB classC classD classE files
frequencies
close all

num_coeffs = 15; % Number of MFCCs/features to use.
down_factor = 1; % Downsampling factor; 1 = no downsampling.

window_length = 1; % FFT Window length normalized to 1 = 1s.
    E.g. 0.5 = 500ms, 0.1 = 100ms
overlap = .5; % The amount of STFT overlap between windows

```

```

% Create empty data structures to be used later.
coeffs = {};
coeffs_unlabel = [];
delta = [];
sample_lengths = 1;
sample_labels = [];

coeffsA = [];
coeffsB = [];
coeffsC = [];
coeffsD = [];
coeffsE = [];

% Iterate through all of the 90 audio files
% 1. Subtract signal mean
% 2. Take STFT
% 3. Apply triangular filterbank to spectrum, 'S'.
% 4. Extract MFCCs.
% 5. Place MFCCs in appropriate struct, based on class.
for i = 1:90
    class = files(i).class;
    FS = audioinfo(files(i).pathname).SampleRate; % Get
        sampling freq,
    NFFT = FS; % Number of FFT points per window =
        sampling freq. = 52734Hz.
    %audioin = decimate(files(i).audio-mean(files(i).
        audio),down_factor);
    %Optional - used for downsampling.
    audioin = files(i).audio-mean(files(i).audio);
    % Take STFT using a 1 second hamming window and 50%
        overlap.
    [S,~,t] = stft(audioin,

```

```

UU
...
    round(FS/1),...
    Window=hamming(round(NFFT*window_length/
        down_factor),'periodic'),...
    OverlapLength=round(overlap*NFFT/down_factor*
        window_length),...
    FrequencyRange="onesided");
% Create a triangular filter bank; the length
    changes based on the
% length of each audio file; hence we need to
    recreate it for each
% audio file individually.
[filter,cf] = designAuditoryFilterBank(FS/
    down_factor/2,FFTLength=round(NFFT*window_length/
        down_factor),NumBands=num_coeffs);
spectrum = filter*(S.*conj(S));
ceptr_coeffs = cepstralCoefficients(spectrum,
    NumCoeffs=size(filter,1));
%delta_coeffs = audioDelta(ceptr_coeffs);
% Optional - for testint delta/delta-delta feature
    performance.

% The following code combines features together to
    create the
% entire database of MFCCs for all audio files.
coeffs_unlabel = vertcat(coeffs_unlabel,ceptr_coeffs)
    ;

% This code adds the class label to the last column
    of each MFCC
% feature matrix so that now we have the feature
    vectors (columns)

```

```

% and they are all appropriately labeled. The second
    to last column
% is always the 'class' and the last column is the
    file ID number -
% for keeping track of which specific audio file the
    data is.
[r,~] = size(cept_coeffs);
labels = string(cellstr(repmat(files(i).class,r,1)))
    ;
labels_ID = repmat(files(i).ID,r,1);
cept_coeffs_label = horzcat(num2cell(cept_coeffs),
    cellstr(labels),num2cell(labels_ID));
files(i).coeffs = cept_coeffs_label;

sample_lengths = [sample_lengths; r];
sample_labels = [sample_labels; cellstr(files(i).
    class)];
coeffs = vertcat(coeffs,cept_coeffs_label);

% This code creates datasets of the features for
    each specific
% class.

switch class
case 'A'
    coeffsA = [coeffsA; cell2mat(files(i).coeffs
        (:,1:num_coeffs))];
case 'B'
    coeffsB = [coeffsB; cell2mat(files(i).coeffs
        (:,1:num_coeffs))];
case 'C'
    coeffsC = [coeffsC; cell2mat(files(i).coeffs
        (:,1:num_coeffs))];

```

```

        case 'D'
            coeffsD = [coeffsD; cell2mat(files(i).coeffs
                (:,1:num_coeffs))];
        case 'E'
            coeffsE = [coeffsE; cell2mat(files(i).coeffs
                (:,1:num_coeffs))];
    end
end

% 'Data' is the complete set of MFCCs features. The first M
% columns are the
% data, where M is the number of features used/number of
% filters in the
% triangular filter bank. The second to last column is the
% class label, the
% last column is the specific file ID label.

% Useful to have the data in table format.
Data = cell2table(coeffs);

% Plot the triangular filterbank.
f = linspace(0,FS/down_factor/2,length(filter));
figure;
hold off;
plot(f,filter);
grid on;
Ftitle = title('Triangular Filter Bank');
Fxlabel = xlabel('Frequency (Hz)');
Fylabel = ylabel('Amplitude');
set([Ftitle, Fxlabel, Fylabel], 'FontName', 'Times New Roman
    ');
set([Fxlabel, Fylabel], 'FontSize', 12);

```

```

set(Ftitle, 'FontSize', 12, 'FontWeight' , 'bold');
set(gca, 'FontName', 'Times New Roman');
set(gca, 'FontSize', 12);

% Plot the MFCCs for one class (A in this example).
figure;mesh(coeffsA);
Ftitle = title('Class A Mel-frequency Cepstrum Coefficients'
    );
Fxlabel = xlabel('Mel-frequency Cepstrum Coefficient Number'
    );
Fylabel = ylabel('Observation Number');
Fzlabel = zlabel('Amplitude');
set([Ftitle, Fxlabel, Fylabel,Fzlabel], 'FontName', 'Times
    New Roman');
set([Fxlabel, Fylabel, Fzlabel], 'FontSize', 12);
set(Ftitle, 'FontSize', 12, 'FontWeight' , 'bold');

% Plot the periodogram for one signal.
[pxx,f] = periodogram(audioin,[],length(audioin),FS,'
    onesided');
figure; plot(f(1:round(length(pxx)/20)),10*log10(pxx(1:round
    (length(pxx)/20))));
Ftitle = title('Periodogram Power Spectral Density Estimate'
    );
Fxlabel = xlabel('Frequency (Hz)')
Fylabel = ylabel('Power/Frequency (dB/Hz)')
set(gca, 'FontName', 'Times New Roman');
set(gca, 'FontSize', 12);

format shortg
number_mfccs = [1 2 3 5 10 20 40];

```

```

% The for-loop here is to test with different numbers of
    features.
% For each loop, it performs training and testing using k =
    5 folds
% cross-validation.

% This for loop may be modified to test how different
    parameters affect
% overall classifier performance. For this case, it is being
    used to test
% the number of features.
for loop = 1:length(number_mfccs)
    num = number_mfccs(loop);
    kfolds = 5;
    % 'cvpartition' creates 5 partitions of the data which
        will be used for training
    % and validation; 4 for training and 1 for validation.
    c = cvpartition(table2array(Data(:,num_coeffs+2)), 'Kfold
        ',kfolds, 'Stratify',true);

    % Create empty arrays to be used to hold the performance
        data. The
    % first acronym denotes the type of performance metric.
        The second
    % denotes for which ML algo it is used: GMM = Gaussian
        mixture model.
    % RF = random forest. KNN = K-nearest neighbors.
    % mcrv = minimum cross-validation classification rate
        vector
    % pv = precision vector.
    % rv = recall vector.

```

```

mcrv_gmm = [];
mcrv_rf = [];
mcrv_knn = [];

pv_gmm = [];
pv_rf = [];
pv_knn = [];

rv_gmm = [];
rv_rf = [];
rv_knn = [];

% iterate 5 times, once for each k = 5 folds.
for i = 1:kfolds
    i
    Test_idx = c.test(i); % Get the testing indices of
        the partitioned data.
    Train_idx = c.training(i); % Get the training
        indices of the partitioned data.
    % Extract the training data.
    training = [Data(Train_idx,1:num),Data(Train_idx,
        num_coeffs+1:num_coeffs+2)];
    % Extract the testing data.
    testing = [Data(Test_idx,1:num),Data(Test_idx,
        num_coeffs+1:num_coeffs+2)];
    % Call the algorithm function; a user defined
        function which
    % handles all of the training and testing for all of
        the
    % algorithms.
    [mcr_gmm,mcr_rf,mcr_knn,p_gmm,p_rf,p_knn,r_gmm,r_rf,
        r_knn]=ALGO(training, testing);

    % Get the performance results back for each loop and

```

```

        put them in
    % array to be used later to get the overall
        performance of each
    % classifier.
    mcrv_gmm(i) = mcr_gmm;
    mcrv_rf(i) = mcr_rf;
    mcrv_knn(i) = mcr_knn;

    pv_gmm(i) = p_gmm;
    pv_rf(i) = p_rf;
    pv_knn(i) = p_knn;

    rv_gmm(i) = r_gmm;
    rv_rf(i) = r_rf;
    rv_knn(i) = r_knn;

end

disp(['GMM K-folds cross validation rate ' num2str(round(
    (mean(mcrv_gmm),2)) '%')]);
disp(['RF K-folds cross validation rate ' num2str(round(
    mean(mcrv_rf),2)) '%')]);
disp(['KNN K-folds cross validation rate ' num2str(round(
    (mean(mcrv_knn),2)) '%')]);

% These are the performance results for each loop of 5
    iterations.
gmm_classrate(loop) = round(mean(mcrv_gmm),3);
gmm_cr_er(loop) = std(mcrv_gmm);

rf_classrate(loop) = round(mean(mcrv_rf),3);

```

```

rf_cr_er(loop) = std(mcrv_rf);

knn_classrate(loop) = round(mean(mcrv_knn),3);
knn_cr_er(loop) = std(mcrv_knn);

gmm_prec(loop) = round(mean(pv_gmm),3);
gmm_prec_er(loop) = std(pv_gmm);

rf_prec(loop) = round(mean(pv_rf),3);
rf_prec_er(loop) = std(pv_rf);

knn_prec(loop) = round(mean(pv_knn),3);
knn_prec_er(loop) = std(pv_knn);

gmm_recall(loop) = round(mean(rv_gmm),3);
gmm_recall_er(loop) = std(rv_gmm);

rf_recall(loop) = round(mean(rv_rf),3);
rf_recall_er(loop) = std(rv_rf);

knn_recall(loop) = round(mean(rv_knn),3);
knn_recall_er(loop) = std(rv_knn);

end

% Plot the classificaion rate results for each classifier
% based on
% whatever variable is changing; in this case, the number of
% MFCCs used.
figure;

ax = axes();
g1 = plot(ax,number_mfccs, gmm_classrate,'Color','Blue');

```

```

grid on;
hold on;
hE1 = errorbar(ax,number_mfccs, gmm_classrate, gmm_cr_er, '
    HandleVisibility','off','CapSize',12);
g2 = plot(ax,number_mfccs, rf_classrate,'Color','Green');
hE2 = errorbar(ax,number_mfccs, rf_classrate, rf_cr_er, '
    HandleVisibility','off','CapSize',12);
g3 = plot(ax,number_mfccs, knn_classrate,'Color','Red');
hE3 = errorbar(ax,number_mfccs, knn_classrate, knn_cr_er, '
    HandleVisibility','off','CapSize',12);
hLegend = legend('GMM','RF','KNN');
hXLabel = xlabel('Number of MFCCs Used');
hYLabel = ylabel('Classification Rate (%)');
hTitle = title('Classification Rate');

set(g1, 'Color', 'Blue');
set(hE1, 'Color', 'Blue');
set(g2, 'Color', 'Green');
set(hE2, 'Color', 'Green');
set(g3, 'Color', 'Red');
set(hE3, 'Color', 'Red');
set(hE1, 'LineStyle', 'none', 'Marker', '.', 'Color', 'Blue'
    );
set(hE2, 'LineStyle', 'none', 'Marker', '.', 'Color', 'Green'
    );
set(hE3, 'LineStyle', 'none', 'Marker', '.', 'Color', 'Red')
    ;

set([g1,g2,g3], 'LineWidth', 2);
set([hE1,hE2,hE3], 'LineWidth', 1);

set(gca, 'FontName', 'Times New Roman');
set([hTitle, hXLabel, hYLabel], 'FontName', 'Times New Roman

```

```

    ');
set([hLegend, gca], 'FontSize', 10);
set([hXLabel, hYLabel], 'FontSize', 12);
set(hTitle, 'FontSize', 12, 'FontWeight' , 'bold');

% Plot the 'Precision' results for each classifier based on
% whatever
% variable is changing; in this case, the number of MFCCs
% used.
figure;
ax = axes();
g1 = plot(ax,number_mfccs, gmm_prec,'Color','Blue');
grid on;
hold on;
hE1 = errorbar(ax,number_mfccs, gmm_prec, gmm_prec_er, '
    HandleVisibility','off','CapSize',12);
g2 = plot(ax,number_mfccs, rf_prec,'Color','Green');
hE2 = errorbar(ax,number_mfccs, rf_prec, rf_prec_er, '
    HandleVisibility','off','CapSize',12);
g3 = plot(ax,number_mfccs, knn_prec,'Color','Red');
hE3 = errorbar(ax,number_mfccs, knn_prec, knn_prec_er, '
    HandleVisibility','off','CapSize',12);
hLegend = legend('GMM','RF','KNN');
hXLabel = xlabel('Number of MFCCs Used');
hYLabel = ylabel('Macro-averaged Precision (%)');
hTitle = title('Precision');

set(g1, 'Color', 'Blue');
set(hE1, 'Color', 'Blue');
set(g2, 'Color', 'Green');
set(hE2, 'Color', 'Green');

```

```

set(g3, 'Color', 'Red');
set(hE3, 'Color', 'Red');
set(hE1, 'LineStyle', 'none', 'Marker', '.', 'Color', 'Blue'
    );
set(hE2, 'LineStyle', 'none', 'Marker', '.', 'Color', 'Green'
    );
set(hE3, 'LineStyle', 'none', 'Marker', '.', 'Color', 'Red')
    ;

set([g1,g2,g3], 'LineWidth', 2);
set([hE1,hE2,hE3], 'LineWidth', 1);

set(gca, 'FontName', 'Times New Roman');
set([hTitle, hXLabel, hYLabel], 'FontName', 'Times New Roman'
    );
set([hLegend, gca], 'FontSize', 10);
set([hXLabel, hYLabel], 'FontSize', 12);
set(hTitle, 'FontSize', 12, 'FontWeight' , 'bold');

% Plot the 'Recall' results for each classifier based on
% whatever variable
% is changing; in this case, the number of MFCCs used.
figure;
ax = axes();
g1 = plot(ax,number_mfccs, gmm_recall,'Color','Blue');
grid on;
hold on;
hE1 = errorbar(ax,number_mfccs, gmm_recall, gmm_recall_er, '
    HandleVisibility','off','CapSize',12);
g2 = plot(ax,number_mfccs, rf_recall,'Color','Green');
hE2 = errorbar(ax,number_mfccs, rf_recall, rf_recall_er, '
    HandleVisibility','off','CapSize',12);

```

```

g3 = plot(ax,number_mfccs, knn_recall,'Color','Red');
hE3 = errorbar(ax,number_mfccs, knn_recall, knn_recall_er, '
    HandleVisibility','off','CapSize',12);
hLegend = legend('GMM','RF','KNN');
hXLabel = xlabel('Number of MFCCs Used');
hYLabel = ylabel('Macro-averaged Recall (%)');
hTitle = title('Recall');

set(g1, 'Color', 'Blue');
set(hE1, 'Color', 'Blue');
set(g2, 'Color', 'Green');
set(hE2, 'Color', 'Green');
set(g3, 'Color', 'Red');
set(hE3, 'Color', 'Red');
set(hE1, 'LineStyle', 'none', 'Marker', '.', 'Color', 'Blue'
    );
set(hE2, 'LineStyle', 'none', 'Marker', '.', 'Color', 'Green'
    );
set(hE3, 'LineStyle', 'none', 'Marker', '.', 'Color', 'Red')
    ;

set([g1,g2,g3], 'LineWidth', 2);
set([hE1,hE2,hE3], 'LineWidth', 1);

set(gca, 'FontName', 'Times New Roman');
set([hTitle, hXLabel, hYLabel], 'FontName', 'Times New Roman'
    );
set([hLegend, gca], 'FontSize', 10);
set([hXLabel, hYLabel], 'FontSize', 12);
set(hTitle, 'FontSize', 12, 'FontWeight', 'bold');

```

```

%
-----

% This function takes as input a training and testing data
% set (like the
% ones created using 'cvpartition' and outputs all of the
% relevant
% performance metrics such as: classification rate,
% precision, and recall)
% for each algorithm used in this study: GMMs, RF, and KNN.
function [mcr_gmm,mcr_rf,mcr_knn,p_gmm,p_rf,p_knn,r_gmm,r_rf
,r_knn] = ALGO(train,test)
    % Important to randomize the RNG each time we call this
    % function
    rng('shuffle')
    options = statset('MaxIter',600);
    % Number of Gaussian distributions per GMM.
    num_gauss = [6 6 6 6 6];
    % Extract the list of file IDs - to be used later.
    ID_list = unique(table2array(train(:,size(train,2))));
    % Simple class labels
    num_class = ['A','B','C','D','E'];

    % Extract from the training set ALL data which has the
    % label 'A' - so
    % that we may train a GMM only on the data which is
    % labeled as 'A'
    % Repeat this for each class A, B, C, D, E.
    [r,~] = find(char(table2array(train(:,size(train,2)-1)))
    == 'A');
    Atrain = table2array(train(r,1:size(train,2)-2));

```

```

[r,~] = find(char(table2array(train(:,size(train,2)-1)))
    == 'B');
Btrain = table2array(train(r,1:size(train,2)-2));

[r,~] = find(char(table2array(train(:,size(train,2)-1)))
    == 'C');
Ctrain = table2array(train(r,1:size(train,2)-2));

[r,~] = find(char(table2array(train(:,size(train,2)-1)))
    == 'D');
Dtrain = table2array(train(r,1:size(train,2)-2));

[r,~] = find(char(table2array(train(:,size(train,2)-1)))
    == 'E');
Etrain = table2array(train(r,1:size(train,2)-2));

% Create GMMs for each set of data based on class (e.g.
    Atrain, Btrain,
% etc.)
% gmm_size is the total number of training points used
    to create that
% GMM. It will be used later to calculate the a priori
    probability of
% an individual data point belonging to a specific class
.
gmm_size = [length(Atrain) length(Btrain) length(Ctrain)
    length(Dtrain) length(Etrain)];
gmmA = fitgmdist(Atrain,num_gauss(1),'Options',options,'
    SharedCovariance',true);
gmmB = fitgmdist(Btrain,num_gauss(2),'Options',options,'
    SharedCovariance',true);
gmmC = fitgmdist(Ctrain,num_gauss(3),'Options',options,'
    SharedCovariance',true);

```

```

gmmD = fitgmdist(Dtrain,num_gauss(4),'Options',options,'
    SharedCovariance',true);
gmmE = fitgmdist(Etrain,num_gauss(5),'Options',options,'
    SharedCovariance',true);
gmm = {gmmA,gmmB,gmmC,gmmD,gmmE};

% Create the KNN model using the training data.
knn_model = fitcknn(train(:,1:size(train,2)-2),train(:,
    size(train,2)-1),'NumNeighbors',10,'Standardize',1);
% Get the predictions from the KNN model using the test
    data.
knn_predict = predict(knn_model,test(:,1:size(train,2)
    -2));
knn_predict = cell2mat(knn_predict);

% Create the RF model using the training data.
Mdl13 = TreeBagger(500,train(:,1:size(train,2)-2),train
    (:,size(train,2)-1),'OOBPrediction','On',...
'Method','classification');
% Get the predictions of the RF model using the test
    data set.
yfit_rf = cell2mat(predict(Mdl13,test(:,1:size(test,2)
    -2)));

% Create empty arrays to be populated in the next
    section.
pred_rf = zeros(1,90);
pred_knn = zeros(1,90);
RF = zeros(1,90);
KNN = zeros(1,90);
GMM = zeros(1,90);
pred_gmm = zeros(1,90);
g1 = blanks(90);

```

```

for i = 1:90
    ID = ID_list(i);
    % Extract all samples corresponding to file ID
    [r,~] = find(table2array(test(:,size(test,2)))==ID);

    % For RF, count the number of occurrences of the
        predicted class.
    [~, gg, a] = groupcounts(yfit_rf(r));
    % The predicted class corresponds to the highest
        percentage
    % (majority).
    [~,k] = max(a);
    pred_rf(i) = gg(k);
    % Put a '1' in RF if the predicted class matches the
        actual class,
    % otherwise, put a '0'. Now we have the recombined
        predictions on a
    % per-audio file bases.
    RF(i) = gg(k) == unique(char(table2array(test(r,size
        (test,2)-1)))));

    % For KNN, count the number of occurrences of the
        predicted class.
    [~, g, a] = groupcounts(knn_predict(r));
    % The predicted class corresponds to the highest
        percentage
    % (majority).
    [~,k] = max(a);
    pred_knn(i) = g(k);
    % Put a '1' in KNN if the predicted class matches
        the actual class,
    % otherwise, put a '0'. Now we have the recombined
        predictions on a

```

```

% per-audio file bases.
KNN(i) = g(k) == unique(char(table2array(test(r,size
    (test,2)-1)))));

% For GMM, we call the pdf function which takes as
    input a GMM
% model file, and the second argument is the testing
    data.
% The output is multiplied by the probability that a
    point of data
% has to belong to that class 'PCk'. The result
    after this
% multiplication is the a posteriori probability of
    a data point
% belonging to that class ('probs'). We iterate 5
    times; once for
% each GMM. The final output will be a percentage
    for each GMM. We
% select the GMM with the highest percentage as the
    final predicted
% class.
probs = [];
PCk = [gmm_size(1)/size(train,1) gmm_size(2)/size(
    train,1) gmm_size(3)/size(train,1) gmm_size(4)/
    size(train,1) gmm_size(5)/size(train,1)];
for j = 1:5
    P = pdf(gmm{j},table2array(test(r,1:size(test,2)
        -2)));
    probs(:,j) = P*PCk(j);
end
probs = probs./sum(probs,2);
if length(r) == 1
    h = probs;

```

```

else
    h = round(sum(probs/length(r))*100,2);
end
[~, I] = max(sum(probs));
% In the same way as the other ML algorithms, we
    give GMM a value
% of '1' if it predicted the correct class for that
    specific audio
% file. Otherwise we give it '0'.
GMM(i) = num_class(I) == unique(char(table2array(
    test(r,size(test,2)-1))));
pred_gmm(i) = char(num_class(I));
% g1 is a vector of the "actual" class for each
    audio file -
% basically it is the truth - will be used to create
    confusion
% matrices.
g1(i) = unique(char(table2array(test(r,size(test,2)
    -1))));
end
% Get the classification rate and make it in percentage
    form.
mcr_gmm = sum(GMM)/90*100;
mcr_knn = sum(KNN)/90*100;
mcr_rf = sum(RF)/90*100;
% g1 is the 'truth' vector = all of the correct classes.
g1 = g1';
% g2_gmm is the vector of GMM model's predictions.
g2_gmm = char(pred_gmm');
% g2_knn is the vector of KNN model's predictions.
g2_knn = char(pred_knn');
% g2_rf is the vector of RF model's predictions.
g2_rf = char(pred_rf');

```

```

% Create confusion matrices for each model.
cm = confusionchart(g1,g2_gmm);

% Calculate precision and recall for each model using
    macro-averaging.
r_gmm = diag(cm.NormalizedValues)'./sum(cm.
    NormalizedValues,1);

% Sometimes 'NaN's show up in the previous calculation
    because of 0/0.
% We set these equal to 0 in order to calculate
    precision and recall.
r_gmm(isnan(r_gmm))=0;
r_gmm = mean(r_gmm)*100;

p_gmm = diag(cm.NormalizedValues)./sum(cm.
    NormalizedValues,2);

% Sometimes 'NaN's show up in the previous calculation
    because of 0/0.
% We set these equal to 0 in order to calculate
    precision and recall.
p_gmm(isnan(p_gmm))=0;
p_gmm = mean(p_gmm)*100;

cm = confusionchart(g1,g2_knn);
r_knn = mean(diag(cm.NormalizedValues)'./sum(cm.
    NormalizedValues,1))*100;
p_knn = mean(diag(cm.NormalizedValues)./sum(cm.
    NormalizedValues,2))*100;

cm = confusionchart(g1,g2_rf);

```

```
r_rf = mean(diag(cm.NormalizedValues)'./sum(cm.  
    NormalizedValues,1))*100;  
p_rf = mean(diag(cm.NormalizedValues)./sum(cm.  
    NormalizedValues,2))*100;  
  
end
```

APPENDIX B: Confusion Matrices

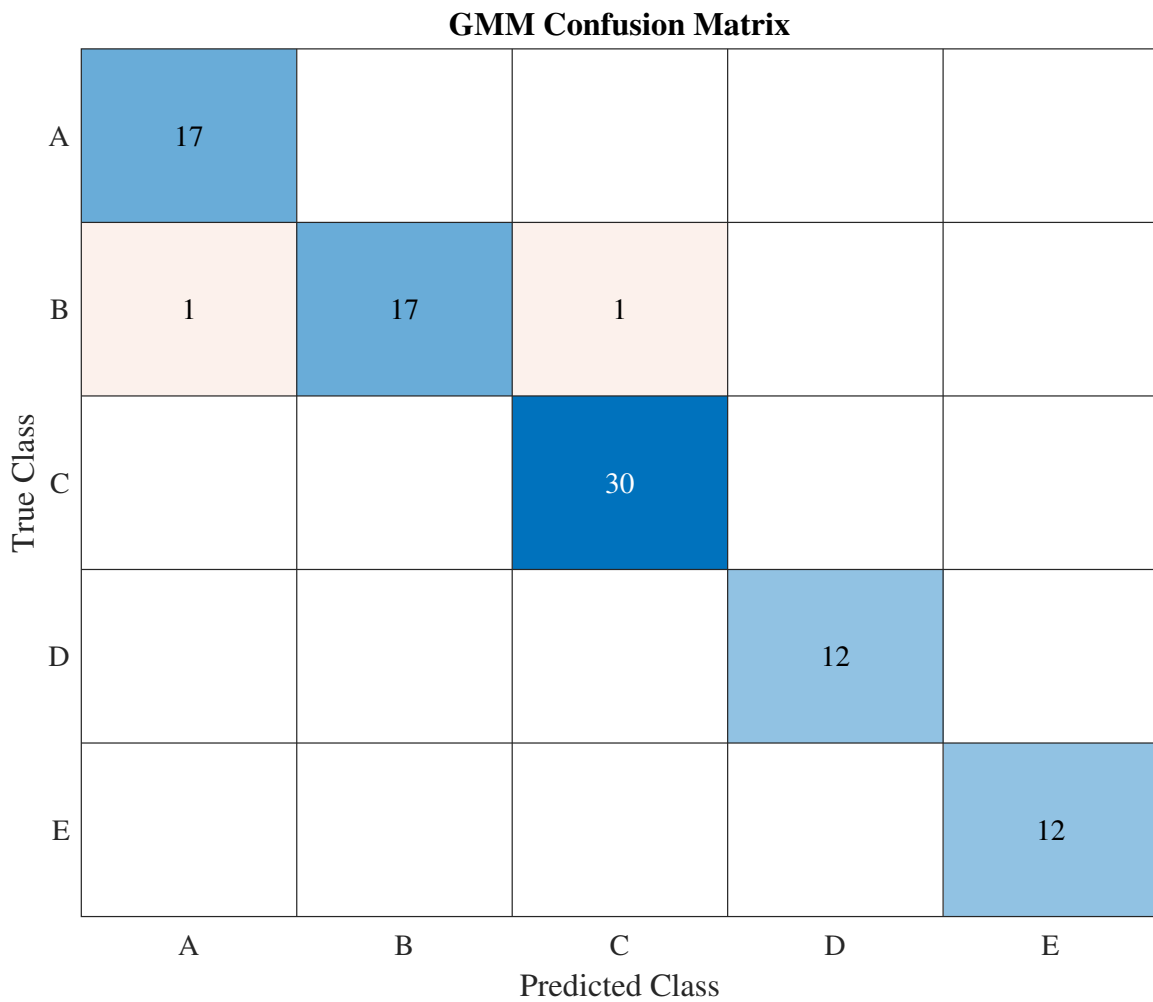


Figure B.1. GMM Confusion Matrix.

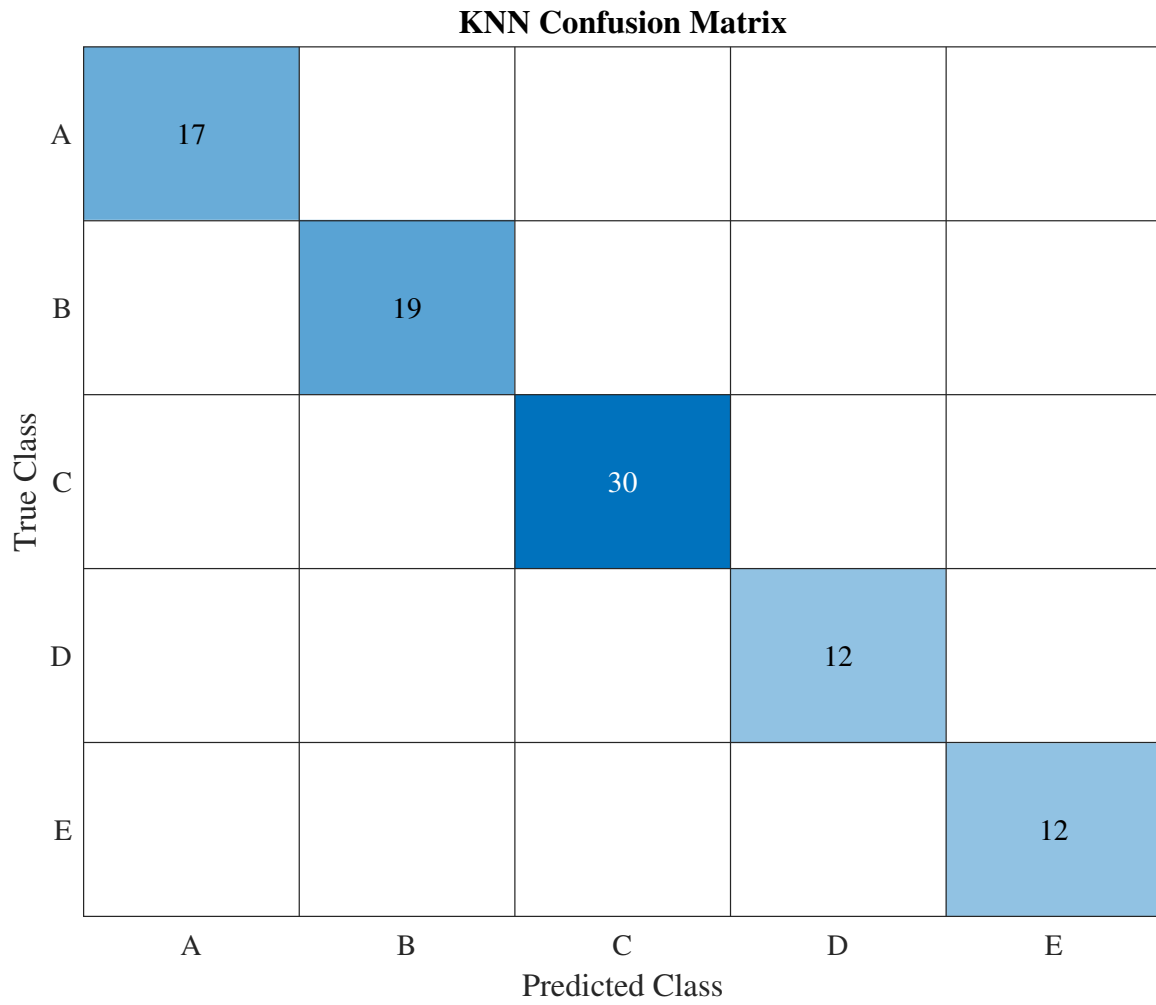


Figure B.2. KNN Confusion Matrix.

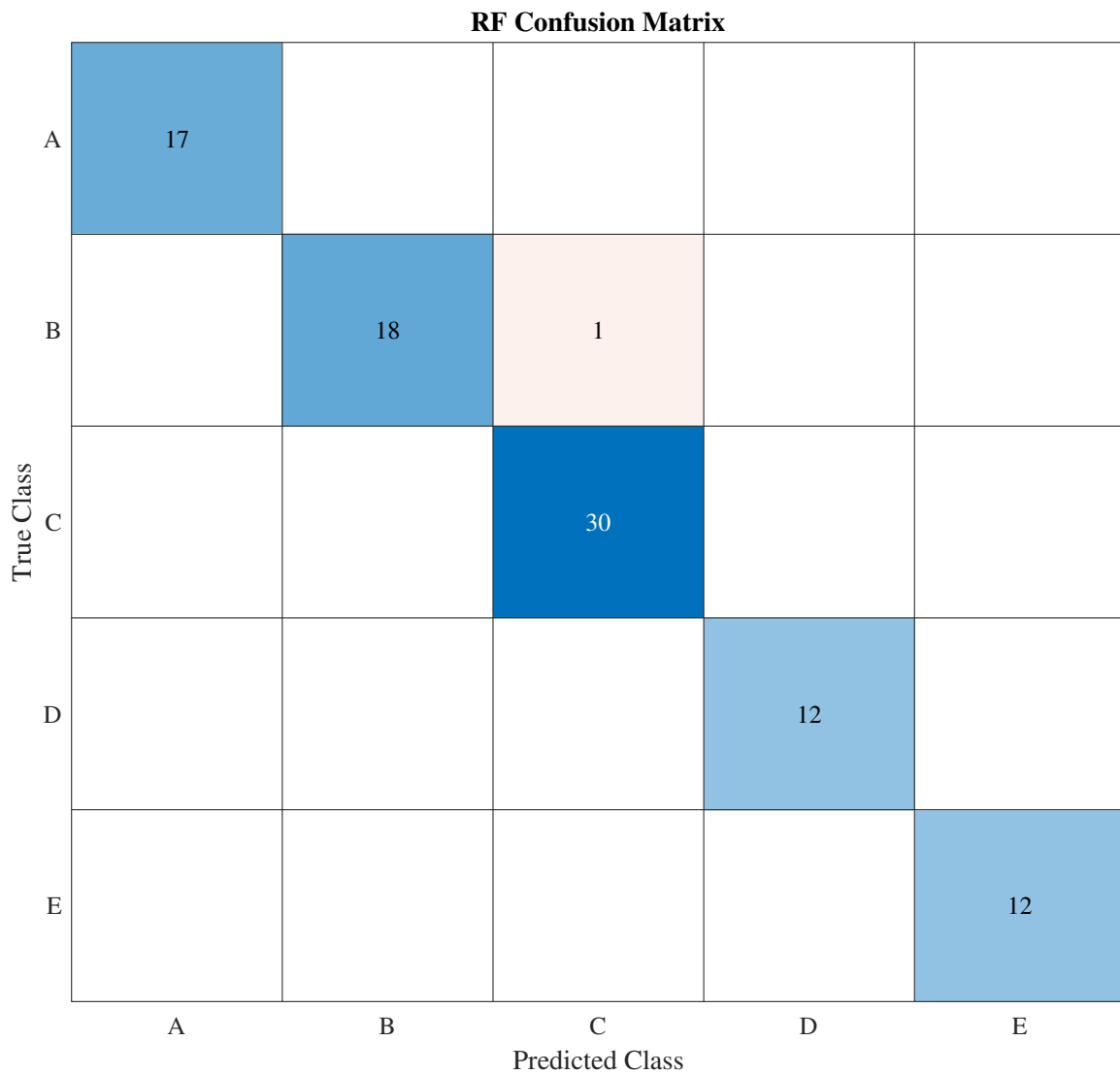


Figure B.3. RF Confusion Matrix.

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

- [1] J. Liu, Y. He, Z. Liu, and Y. Xiong, “Underwater target recognition based on line spectrum and support vector machine,” in *2014 International Conference on Mechatronics, Control and Electronic Engineering (MCE-14)*. Atlantis Press, March 2014, pp. 79–84. Available: <https://www.atlantis-press.com/proceedings/mce-14/14065>
- [2] A. Das, A. Kumar, and R. Bahl, “Marine vessel classification based on passive sonar data: the cepstrum-based approach,” *IET Radar, Sonar & Navigation*, vol. 7, no. 1, pp. 87–93, January 2013. Available: <https://onlinelibrary.wiley.com/doi/10.1049/iet-rsn.2011.0142>
- [3] D. Santos-Domínguez, S. Torres-Guijarro, A. Cardenal-López, and A. Pena-Gimenez, “ShipsEar: An underwater vessel noise database,” *Applied Acoustics*, vol. 113, pp. 64–69, December 2016.
- [4] X. Luo and Y. Feng, “An underwater acoustic target recognition method based on restricted boltzmann machine,” *Sensors*, vol. 20, no. 18, p. 5399, September 2020. Available: <https://www.mdpi.com/1424-8220/20/18/5399>
- [5] X. Ke, F. Yuan, and E. Cheng, “Underwater acoustic target recognition based on supervised feature-separation algorithm,” *Sensors*, vol. 18, no. 12, p. 4318, December 2018. Available: <https://www.mdpi.com/1424-8220/18/12/4318>
- [6] D. Santos-Domínguez, S. Torres-Guijarro, A. Cardenal-López, and A. Pena-Gimenez, “Shipsear database,” November 2014, accessed May 10, 2023. Available: <http://atlanttic.uvigo.es/underwaternoise/indexDB.php>
- [7] I. Nabney, *NETLAB: Algorithms for Pattern Recognition* (Advances In Pattern Recognition). New York, NY, USA: Springer, 2002.
- [8] A. P. Dempster, N. M. Laird, and D. B. Rubin, “Maximum likelihood from incomplete data via the EM algorithm,” *Journal of the Royal Statistical Society. Series B*, vol. 39, no. 1, pp. 1–38, 1977.
- [9] D. Arthur and S. Vassilvitskii, “K-means++: The advantages of careful seeding,” *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, vol. 8, pp. 1027–1035, January 2007.
- [10] Chire, “Em clustering of old faithful eruption data,” August 2012, accessed April 21, 2023. Available: https://upload.wikimedia.org/wikipedia/commons/6/69/EM_Clustering_of_Old_Faithful_data.gif

- [11] T. Cover and P. Hart, “Nearest neighbor pattern classification,” *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, January 1967.
- [12] S. Jaiswal, “K-nearest neighbor(knn) algorithm for machine learning,” May 2018, accessed April 21, 2023. Available: <https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning>
- [13] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, October 2001.
- [14] AlgoDaily, “Getting to know decision trees,” April 2023, accessed June 19, 2023. Available: <https://algodaily.com/lessons/decision-trees-basics>
- [15] Sirakorn, “Ensemble bagging,” January 2020, accessed May 10, 2023. Available: https://upload.wikimedia.org/wikipedia/commons/c/c8/Ensemble_Bagging.svg
- [16] Brownlee, “A gentle introduction to k-fold cross-validation,” May 2018, accessed April 21, 2023. Available: <https://machinelearningmastery.com/k-fold-cross-validation/>
- [17] Z. Shen, “3 min of machine learning: Cross validation,” March 2020, accessed April 21, 2023. Available: https://zitaoshen.rbind.io/project/machine_learning/machine-learning-101-cross-validation/
- [18] T. M. Inc., *MATLAB version: 9.13.0 (R2022b) Update 1*, Natick, Massachusetts, United States, 2022. Available: <https://www.mathworks.com>
- [19] A. Amini, “MIT introduction to deep learning 6.s191: Lecture 4,” lecture, Introduction to Deep Learning 6.S191, March 31, 2023, Massachusetts Institute of Technology, March 2023, accessed June 29, 2023. Available: <http://introtodeeplearning.com/>

Initial Distribution List

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California



DUDLEY KNOX LIBRARY

NAVAL POSTGRADUATE SCHOOL

WWW.NPS.EDU

WHERE SCIENCE MEETS THE ART OF WARFARE