Listen to Your Heart: Feature Extraction and Classification Methods for Heart Sounds

Angela Chao*, Shirley Ng† and Linda Wang‡
Department of Systems Design Engineering
University of Waterloo
Waterloo, Canada
Email: *ajchao@uwaterloo.ca, †s25ng@uwaterloo.ca, ‡ly8wang@uwaterloo.ca

Abstract—**Heart irregularities are commonly detected using a stethoscope by a physician. Currently, there are digital stethoscopes and mobile devices that anyone can use to record their heart sounds, however, without medical knowledge, will not know if there are any irregularities. This paper presents a process for classifying those audio heart sounds to five most commonly occurring classes: artifact, extra heart sound, extrasystole, murmur and normal heartbeat. The paper also compares the precisions and F-scores of six machine learning models, which include Naive Bayes, Support Vector Machines and Decision Trees. Using the process outlined in this paper, the results are a significant improvement to the state of the art for all classes except for extrasystole and normal heartbeats. The paper also outlines practicality and next steps to improve audio heart sound classification.**

Keywords—**Heartbeat sounds, heartbeat feature extraction, classification of heartbeats**

I. Introduction

Stethoscope detection currently provides one of the first warning signs of heart disease. During this procedure, which usually occurs in ones annual physical examination, physicians manually listen for and identify irregularities in heart sounds that indicate a need for further investigation [1]. Early detection and intervention in heart disease greatly improves the lifespan and the effectiveness of treatment options [2]. For this reason, the benefits of placing the first-detection capability in the hands of the individual is immense. This study aims to use machine learning methods to identify and classify heartbeat sounds from audio collected from digital stethoscopes and mobile devices available to the average consumer.

From the audio heart sounds, 18 features are extracted using both the whole signal and specific locations in a heartbeat. Using the 18 features, various machine learning models are used to classify into different categories. This paper compares the performance of these machine learning models, such as Naive bayes, SVM and Decision Trees, and presents a guide to selecting relevant features and models when classifying real heart audio.

A. Background

Different heart-sounds, resulting from mechanical cardiac events and heard from stethoscope examinations, can be indicators for various cardiac health concerns. Typically, a heartbeat creates two sounds, lub-dub, that can be heard through a stethoscope. These sounds are termed, respectively, S1 and S2 for the first and second heart sounds within one beat. S1 occurs when the mitral and tricuspid valves close at the end of systole, when blood from the atriums fill into the ventricles. S2 is heard when aortic and pulmonic valve leaflets close after diastole, after blood is pushed out from the ventricles [3]. These two sounds make up what a normal heartbeat should sound like.

When abnormalities occur in cardiac physiology, they are oftentimes reflected in heartbeat sounds. A few of the most commonly occuring abnormal sounds, the ones that this study will be identifying and classifying, are heart murmurs, extrasystole, extra heart sounds, and artifact sounds. Murmurs are extra sounds that occur when there is turbulence in blood flow that causes the extra vibrations that can be heard. Physiologically, this turbulence can be caused by abnormal valve events that cause irregularities to the flow of blood, such as regurgitation through the valves [3]. Extrasystole is a premature contraction of the heart that causes an extra sound before S1 and palpitations due to abnormal electrical circuitry within the heart [4]. Extra heart sounds occur after diastole, S2, due to sudden deceleration of blood, caused by abnormalities in cardiac muscle physiology that affects contraction characteristics [3]. Artifacts were also included in this study to evaluate how robust the machine learning models were in recognition of whether or not an audio sample was a of heart sound.

B. Related Works

There are a few studies that have also done work in classifying various heartbeat sounds using machine learning methods. However, the majority has focused on data obtained from phonocardiography (PCG), which is graphical of heart sounds that are obtained via high-fidelity sensors during medical diagnosis, a technology that is not widely available to the average consumer. Using PCG datasets from PhysioNet, to perform classification using various models. The classification of heart sound signals based on AR models was studied by He, achieving an overall accuracy score of 74% [5]. Ryu used convolutional neural networks to determine normality, and does not offer specificity into the type of abnormality if observed [7]. Singh also uses PCG data to study effectiveness of features and various classifiers in distinguishing between normal and murmur heart sounds [6].

This study is based on Peter J Bentleys dataset, containing audio data, for his Heart Sound Challenge, where he has done preliminary analysis that resulted in 77% precision in identifying normal heart sounds [8]. The main difference between this dataset and data obtained from PCG is that PCG is conducted in a professionally controlled environment, using sophisticated technology, that is able to produce clean and relevant signals. The data used for this study is derived from consumer, app-based retrieval of heartbeat audio for the purpose of exploring capabilities of machine intelligence in identifying heart abnormalities through technology available to the hands of the public.

II. Experimental Setup

A. Datasets

Two different datasets are used to verify the performance of the machine learning models. Dataset A has four classes with 120 total samples. The four classes are artifact, extra heart sound, murmur and normal heartbeat. This data was collected from the general public using an iPhone application, iStethoscope Pro [8], which collects 44100 samples of heart sounds per second. Dataset B has fewer classes and more samples, which were collected from a clinical trial in hospitals using a digital stethoscope [9], which collects 4000 samples of heart sounds per second. This dataset consists of three classes and a total of 461 samples, 149 of which are noisy. The three classes are extrasystole, murmur and normal heartbeat. These two different datasets are used to verify the generalizability of the machine learning models, as well as to compare both dataset results with the current state of the art [bentley].

B. Preprocessing

Prior to feature extraction, the audio signals were preprocessed. The preprocessing includes removing audio files that are less than 2 seconds, downsampling, removing high frequency noise and normalizing the data. Files less than 2 seconds are removed because those files capture one or less heartbeats, making it difficult to extract relevant features. For dataset A, a total of 120 samples are used and for dataset B, a total of 407 samples are used.

The data for dataset A is first downsampled by a factor of 5 and then downsampled by a factor of 2 using wavelet decomposition. Due to the high sampling frequency of 44100 Hz for dataset A, downsampling reduces the size of the data while preserving all relevant information. The final sampling frequency of 4410 Hz is still higher than double 600 Hz, which is frequency of murmurs, the highest heart sound frequency for these datasets [8]. It is required that the sampling frequency is greater than twice the frequency of murmurs due to the Nyquist sampling rate. For dataset B, there is no initial downsampling, however, after the wavelet decomposition, the data is downsampled by 2. The final sampling frequency of 2000 Hz is also higher than the highest heart sound frequency of 600 Hz.

Wavelet decomposition with a fourth-level order six Daubechies filter is used to denoise the signal, which proved beneficial from the results of previous papers [8][5]. After the wavelet decomposition, the audio files are normalized to the same length as the shortest file length, which is 14272 for dataset A and 4011 for dataset B. The audio files capture 3 seconds and 2 seconds of heart sounds for dataset A and B respectively. These lengths are sufficient for capturing heart data for feature extraction. Since the heart sound files were recorded manually, the amplitudes for each file are different. Thus, the audio files are normalized to a maximum amplitude of 1 so that the strengths of the recorded signals will not impact the model performance.

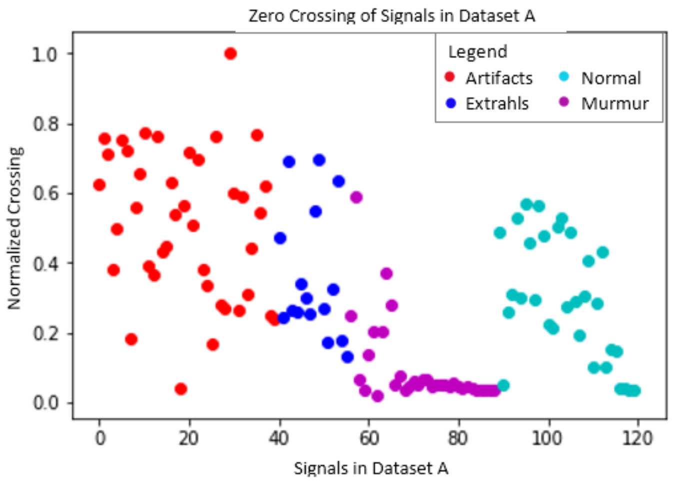

Fig. 1. Normalized zero crossings for dataset A.

III. Feature Extraction

For feature extraction, the signals were analyzed in two groups. For the first group of extractions, the entire signal was analyzed in both the time and frequency domain. The second group uses significant parts of the signal as a whole to extract features. The significant parts of a signal are S1 and S2. A total of 18 different features were extracted, seven of which are based on the entire signal.

The features extracted for the first set of features in the time domain are zero crossings, energy and entropy of energy. In the frequency domain, spectral spread, spectral entropy, spectral flux and Mel Frequency Cepstral Coefficients (MFCCs) were used. All features except MFCCs, were extracted using pyAudioAnalysis, a python library for audio signal analysis. For MFCCs, another python library, python_speech_features, was used.

Zero crossings is the rate of sign changes of the signal during the duration of a particular frame [10]. This captures frequency and intensity information, which was successfully used for speech recognition tasks [11]. Figure III and Figure III illustrates results for dataset A and dataset B zero crossing respectively. The zero crossing for murmurs in dataset A, shown in purple, is the most distinct class. From Figure III, there are no distinct differences between the classes.

Energy is the sum of squares of the signal values, which are normalized by the respective frame length [10]. There

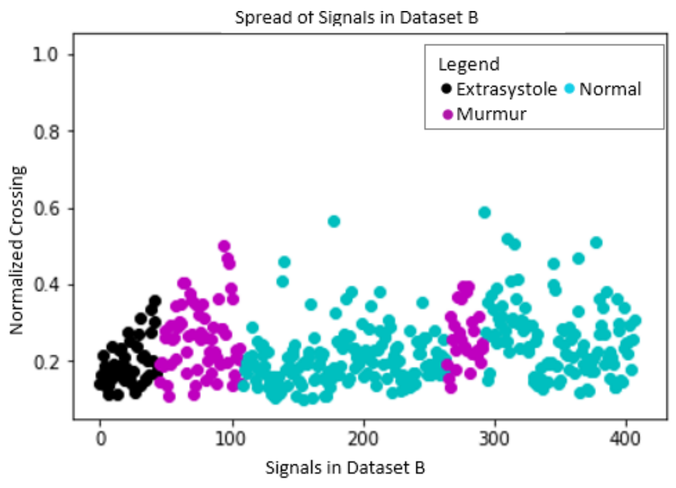

Fig. 2. Normalized zero crossings for dataset B.

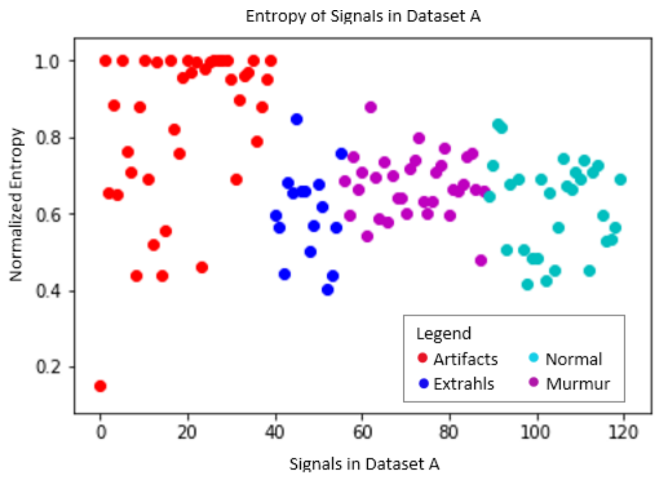

Fig. 3. Normalized entropy of energy for dataset A.

is no apparent distinction between classes using this feature, however, the average value of the murmurs are slightly higher than the other classes. The average value of the normal heartbeats classes are slightly higher than extra heart sounds and extrasystole classes.

The entropy of energy is the entropy of the signals' normalized energies and can be interpreted as a measure of abrupt changes [10]. Although the data is clustered together, the data points from Figure III and III are distributed among the center of each class.

Spectral spread is the second central moment of the spectrum [10]. This shows how the frequency is spread in a given class. From the results, Figure III shows that spectral spread captures features of artifacts, extra heart sounds and murmurs well for dataset A. For dataset B, extrasystoles are clustered within a small range, as shown in Figure III, however, due to the large spread of murmurs and normal heartbeats, some of the data points are in the same range as extrasystoles.

Spectral entropy is the entropy of the normalized spectral energies for the frequency spectrum of a signal [10]. The

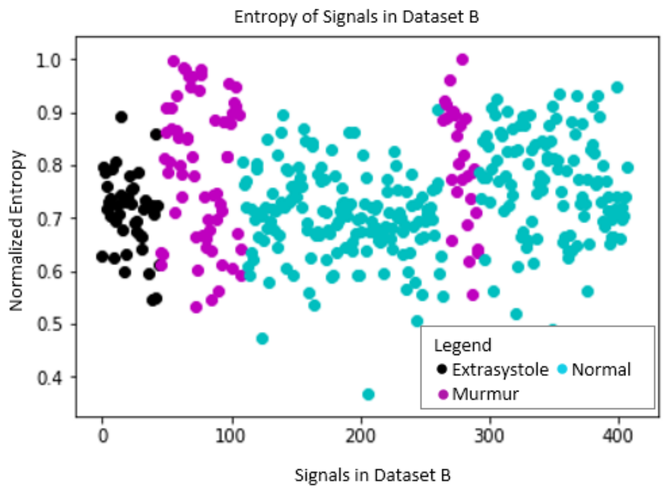

Fig. 4. Normalized entropy of energy for dataset B.

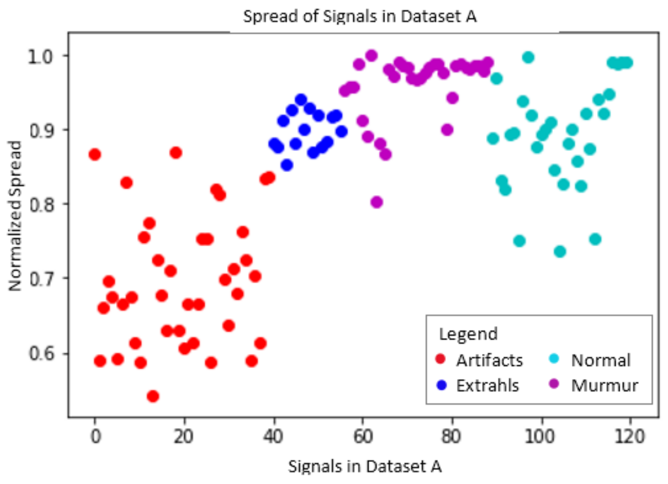

Fig. 5. Normalized spectral spread for dataset A.

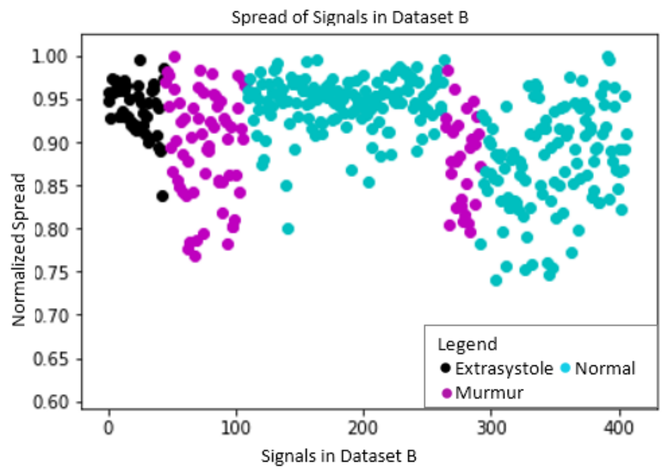

Fig. 6. Normalized spectral spread for dataset B.

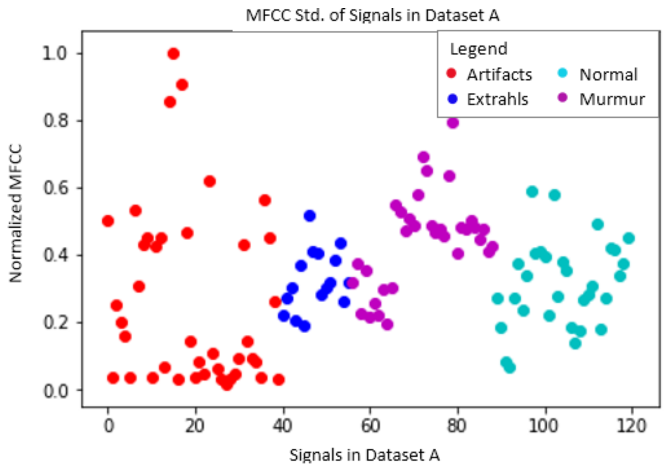

Fig. 7. Normalized MFCC standard deviation for dataset A.

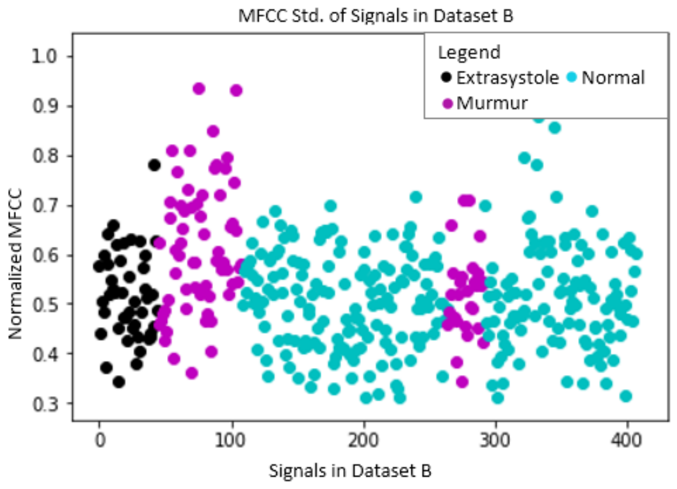

Fig. 8. Normalized MFCC standard deviation for dataset B.

results for dataset A show that artifacts on average have a larger entropy and murmurs have a lower entropy. For dataset B, extrasystole and normal heartbeats have similar entropy ranges, while murmurs and the noisy signals have higher entropy values.

Spectral flux is the squared difference between the normalized magnitudes of the spectra of two successive frames [10]. To get two successive frames, each sample was split in half, with the first half as the first frame and second half as the second frame. The clusters of classes are more evident in dataset A, where each class has a different range of values compared to dataset B.

MFCCs represent where the frequency bands are non-linear but distributed according to Mel-scale [10]. MFCCs are typically used for speech recognition tasks [12][13][14] due to its generation of coefficients being unique for each user [12]. For the feature extraction of heartbeat signals, the top three principal components for the MFCCs are used. Each component also contains one feature vector. The mean and standard deviation is taken of the feature vector as features, which results in a total of six MFCCs features representing each sample. Figure III and III show the standard deviation of the first principal component for dataset A and B. For dataset A, the MFCCs show a distinction between artifacts and heartbeats. For dataset B, the MFCCs are all within a certain range and does not give a clear distinction between classes except that murmurs tend to have more deviations in coefficients.

The second group of features is characterized by the sequence of S1 and S2 in each audio recording. These spikes were found by modifying the existing Python function: find_peaks_cwt() to identify spikes above 15% of the signals normalized amplitude. To distinguish between S1 and S2 peaks, a similar strategy to Bentley and Deng was adapted [8]. Clinical research concluded that the diastolic period between S2 and S1 is normally longer [29]. Thus, the maximum difference between two peaks was declared the diastolic period with the first peak labeled as S2 and the following peak as S1. The starting and ending boundaries of each S1 and S2 were determined based on observation. Different bounds were determined for dataset A and dataset B due to different sampling rates and signal quality.

Six statistical features in the time domain were found according to the S1 and S2 peaks. These features include the time between peaks, standard deviation of the intervals and the maximum to mean ratio of each peak. The duration of the diastolic period was chosen because it has proven to be a consistent predictor of murmurs in current models [18]. To calculate the maximum to mean ratio of each peak, the signals were first normalized and rectified. The maximum value within each S1 and S2 along with the mean value of a peak interval was used for this calculation. Since each heartbeat exhibit different oscillations and amplitudes, the standard deviation within each interval and the deviation of the diastolic period was used to represent the different classes within each dataset.

IV. Methods

Six machine learning models are used in this experiment to classify heartbeats. These machine learning approaches include the Naive Bayesian classifier, support vector machines (SVM), decision trees, Adaboost using the Naive Bayesian classifier, random forest and gradient boosted trees.

A. *Naive Bayes*

Bayesian classifiers are used for a multitude of classification tasks, which range from text classification, such detecting spam [15], to audio classification, such as classifying music [16]. Zhang et al. used Naive Bayes, a technique based on the Bayesian Theorem, to train the detector to decide whether an email is spam or not [15]. In another paper, Fu et al. also used Naive Bayes, but to classify music based on audio features extracted from local windows [16]. Based on the superior performance results achieved by these papers using Bayesian classifiers, this approach is also investigated in this paper to classify audio heartbeats.

Bayesian classifiers are a popular technique in pattern recognition because it is an optimal classifier that minimizes the average probability of error [17]. These classifiers are based on the assumption that prior probabilities and distributions of patterns in the classes are known so that given a pattern, the class label that has the maximum posterior probability is assigned to the given pattern [17]. To calculate the posterior probability, Bayes theorem, shown in Equation 1, is used. Bayes theorem uses the prior probability of the pattern to be classified, $P(y)$, and the likelihood values, $P(x_1, ..., x_n|y)$ to calculate the posterior probability [19].

$$P(y|x_1, ..., x_n) = \frac{P(y)P(x_1, ..., x_n|y)}{P(x_1, ..., x_n)} \tag{1}$$

There are different types of Bayesian classifiers, which adjust the way the class likelihood probabilities are calculated. One type of a Bayesian classifier is the Naive Bayes classifier. This classifier assumes that there is independence between every pair of features, as shown in Equation 2[19]. Using the assumption of conditionally independence between features, the equation that computes the posterior probability is shown in Equation 3. Similar to how different types of Bayesian classifiers adjust the calculation of likelihood probabilities, different forms of Naive Bayes classifiers change the distribution of the likelihood probabilities [19]. For the classification task of heartbeats, the Gaussian Naive Bayes classifier is used based on the results of the class distributions in the feature extractions section. The Gaussian Naive Bayes classifier used in this paper was implemented by Scikit Learn [20]. The prior probabilities of each class are found and inputted as a parameter.

$$P(x_1, ..., x_n|y) = P(x_1|y)P(x_2|y)...P(x_n|y) \tag{2}$$

$$P(y|x_1, ..., x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1, ..., x_n)} \tag{3}$$

B. Support Vector Machines

SVMs are also used for audio classification tasks. Ittichaichareon et al. used SVMs with MFCCs for speech recognition using a limited dataset size and achieved improvement in recognition rates [14]. Li et al. also used SVM to classify content based audio using perceptual and cepstral features [21]. They found that given the feature set, SVMs were able to learn optimal class boundaries between classes. Based on the results from previous works and in addition to SVMs working well with MFCC features, SVMs are also used in this paper.

SVMS are a supervised machine learning method used for classification. A subset of training points, called support vectors, are used for the decision function [22], which also makes this technique memory efficient. Similar to Bayesian Classifiers, SVMs are also versatile as there are different kernel functions that can be specified for decision functions to classify classes in higher dimensions [22]. In this paper, SVM with radial basis function (RBF) kernel and C support vectors are used, which was implemented by Sciki Learn [24]. The C parameter is a penalty for the error term. SVMs with large C values will choose a smaller margin hyperplane if that hyperplane is able to have more correct classifications [23]. To control the RBF kernel, both gamma and the C parameter are used. Gamma defines how much influence a single training example has, where a large gamma has little influence and results in the support vectors only including themselves [23]. For the training, C and gamma are both set to 1 for dataset A and B.

C. Decision Trees

Decision trees are another popular machine learning technique for classification due to its simplicity and interpretability [25]. The method learns simple decision rules based on the features to predict the class of the target [25]. The decision rules for selecting the best attribute to split the tree are based on criterion scores, such a Gini impurity and entropy to measure information gain [26]. Contrary to artificial neural networks, which use a black box model, decision trees use a white box model, where the explanation for a decision can be easily explained with boolean logic [25]. When training the model, decision trees can create over complex trees, which end up overfitting the model. However, overfitting can be minimized by setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree [25].

This method is chosen to classify heartbeats because decisions trees do not need a lot of data, can handle noisy data and can classify classes where its data points could be in multiple clusters in a feature space [27]. For the classification, the decision tree model used was implemented in Scikit Learn [26]. The max depth and number of samples required at a leaf node are set to 5 and 3 respectively to avoid overfitting for dataset A and B.

D. Ensemble Methods

Ensemble methods are also used for this classification task to improve generalizability and robustness over a single estimator, such as Naive Bayes and decision trees. Ensemble methods combine predictions of several base estimators using a learning algorithm in order to improve classification accuracy [31]. There are two main types of common ensemble methods, bootstrap aggregation (bagging) methods and boosting methods. The bagging method used for this classification is random forest and the boosting methods used are AdaBoost and Gradient Tree Boosting, all which were implemented on Scikit Learn [33][32][34].

Bagging methods build several instances of a base estimator, which introduces randomization for the classifier construction, and then aggregates their individual predictions to form a final prediction [31]. Using this method, the variance of the base estimator is reduced. The bagging method chosen is random forest with decision trees as the base estimators. Random forest is a perturb-and-combine technique specifically designed for trees [31][28]. The prediction is then computed as the average prediction of individual classifiers. For training the random forest classifier, the random forest Scikit Learn implementation [33] is used. The number of estimators, max depth and minimum samples at leaf node are set to 100, 4, and 3 respectively for dataset A and 100, 10, and 3 respectively for dataset B.

For the boosting methods, AdaBoost using Naive Bayes as the base estimator and gradient boosted trees are used for the classifiers. The main objective for boosting is to fit a sequence of weak learners on repeatedly modified versions of the data that focuses on examples that are missed by the previous learners [31][30]. To modify versions of data, AdaBoost adjusts the weights. For the first boosting iteration, the weak learner is trained on the original data, with the same weight on each sample. For each successive iteration, the weight is adjusted for each sample and the learning algorithm is applied again on the samples, but this time with adjusted weighted. By adjusting the weights, the learners can focus on samples that are frequently misclassified. The predictions are then made by using majority voting [31][30]. For training the AdaBoost classifier, the AdaBoost Scikit Learn implementation [32] is used. The number of estimators and learning rate are set to 0.001 and 100 for both dataset A and B.

Gradient boosted trees is a generalization of boosting to different loss functions [31]. For each iteration, regression trees are fit on the negative gradient of the loss function [34]. The gradient boosted tree classifier used for this classification was implemented by Scikit Learn [34]. The parameters used are number of estimators, learning rate and maximum depth of the base estimators. These parameter are set to 100, 0.1 and 3 for dataset A and 100, 0.01 and 15 for dataset B.

V. Results

Each method above uses a K-fold cross validation of 4, which means that 25% of the dataset is used for the validation testing and 75% of the dataset is used for training. All final precision and score results listed are an average of the K-fold cross validation.

Precision is one of the measures used to calculate the performance of the machine learning models. Precision estimates the predictive value of a label [sok], for instance, the higher the precision, the more likely the prediction is correct. This means that precision can be used to assess the predictive power of the machine learning model [sok]. Equation 4 shows how the precision is calculated. A prediction is considered true positive (TP) if the predicted classes matches the target class. In the other case, false positive (FP) is if a predicted classed is not the same as the target class. In a confusion matrix with rows being the target class and columns being the predicted class, the false positives for a class are the sum of the values in the corresponding column except the row with the target class.

$$precision = \frac{TP}{TP + FP} \tag{4}$$

F-score is the other measure used. F-score is a composite measure that challenges algorithms with higher recall and measures a classification models usefulness, which is calculated by Equation 6 [sok]. In order to calculate F-score, recall is needed in addition to precision. Recall approximates the probability of a positive label being true [sok] and is shown in Equation 5. This measure assesses the effectiveness of the models used. The false negatives (FN) are how many times the target class is misclassified. In a confusion matrix, the false negatives for a class are the sum of the values in the corresponding row, except the column with the target class. When computing the F-scores, β is set to 1 for equal weighting between precision and recall.

$$recall = \frac{TP}{TP + FN} \tag{5}$$

$$Fscore = \frac{(\beta^2 + 1) * precision * recall}{\beta * precision + recall} \tag{6}$$

Based on the results in Table I and III, SVM, Naive Bayes, AdaBoost and gradient boosted trees have the most consistent results for both datasets. Although decision tree and random forest classifiers have competitive class precision scores for dataset A, these classifiers do not generalize well to dataset B as extrasystole heartbeats are not accurately classified. In Table I and III most or all of the class precisions are better than the state of the art class precisions in Table II and IV [8].

Table I. Precision(%) of each class for dataset A

Class	Naive Bayes	SVM	Decision Trees	Ada-Boost	Random Forest	Gradient Boosting
Artifact	94.72	**97.22**	82.08	95.00	88.75	88.37
Extrahls	42.46	56.25	48.21	41.96	**58.33**	43.33
Murmur	78.48	**91.88**	78.87	75.28	74.79	71.76
Normal	44.05	47.33	42.29	41.79	**50.00**	46.88

Table II. Current state of the art precision(%) of each class for dataset A [8]

Class	Bentley
Artifact	58.33
Extrahls	11.27
Murmur	31.25
Normal	45.83

Table III. Precision(%) of each class for dataset B

Class	Naive Bayes	SVM	Decision Trees	AdaBoost	Random Forest	Gradient Boosting
Extrastole	**18.29**	17.05	0.00	17.18	0.00	17.50
Murmur	47.60	**80.65**	56.05	48.81	70.37	61.09
Normal	72.06	**73.90**	71.00	70.92	71.17	71.25

Table IV. Current state of the art precision(%) of each class for dataset B

Class	Bentley
Extrastole	16.67
Murmur	36.99
Normal	77.67

The F-scores in Table V and VI is a combined measure of precision and recall. Based on these scores, SVM has the best results for dataset A and random forest has the best result for dataset B. However, Table III shows that even though the random forest classifier has high class precisions for murmur and normal heartbeats, it is unable to classify extrasystole heartbeats. This suggests that better features are required for both decision tree and random forest classifiers.

Table V. Overall results for dataset A

Measure	Naive Bayes	SVM	Decision Trees	AdaBoost	Random Forest	Gradient Boosting
Precision	65.83	71.08	62.54	65.85	68.02	64.70
Recall	69.17	71.67	65.00	70.00	71.67	68.33
F-score	67.46	**71.37**	63.75	67.86	69.80	66.47

Table VI. Overall results for dataset B

Measure	Naive Bayes	SVM	Decision Trees	AdaBoost	Random Forest	Gradient Boosting
Precision	44.97	68.55	68.06	46.20	71.26	67.31
Recall	44.97	68.55	68.06	46.20	71.26	67.31
F-score	44.97	68.55	68.06	46.20	**71.26**	67.31

Using a combination of F-scores and individual class precision results, the SVM classifier is found to be the most promising approach given the 18 extracted features. For dataset A, SVM achieved the highest class precisions for artifact and murmurs. The precisions for extra heart sound and normal heartbeats are also competitive with the precisions using random forest. Although the random forest classifier achieved the highest class precision for extra heart sound and normal heartbeats, the precision differences between SVM and random forest shows that SVM classifier achieved the best results with smaller precision differences with the highest class precisions. The precision differences between SVM and random forest for artifact and murmurs are 8.47% and 17.09%, whereas the precision differences for extra heart sound and normal heartbeats are 2.08% and 2.67%. Similarly, for dataset B, SVM achieved the highest class precisions for murmur and normal heartbeats, with only 1.24% behind the highest class precision for extrasystole.

From the results in Table V and VI, SVM classifier also achieved the highest F-score for dataset A and was slightly outperformed by random forest classifier for dataset B. However, since the random forest classifier is not able to classify extrasystole heartbeats, SVM also proves to be the best classifier for dataset B. In addition to SVM classifiers, AdaBoosting with Naive Bayes as base estimator and gradient boosted trees also show promise in the tasks of classifying audio heartbeats, as the results are comparable to the results of SVM.

VI. Discussion

The comparison of techniques and class precision results with the state of the art offer many insights into implications in the medical industry, nature of the datasets and where improvements can be made.

The first insight is the possible implications to the medical industry. When implementing a classifier into real world situations, it is important to doctors and other medical physicians to understand why and how a decision is made. A white box system can also indicate which features are more important and should have more focus. This can help with finding improvements to the system, such as knowing which features should be improved on and if more relevant features are required. As discussed in the methods section for decision trees, one advantage of decision trees is its ability to explain why a decision is made based on its impurity scores and features that the tree chose to split on. Based on the results, gradient boosted trees performed better for most of the class precisions compared to state of the art and is competitive with the results of the SVM classifier. For the purposes of having a white box system, gradient boosted trees hold a lot of promise in its ability to both explain decisions and achieve high precision among techniques and state of the art results.

The second insight, based on the results, is the nature of the datasets and how classifiers perform on unbalanced datasets. The SVM classifier achieved the best results because it is able to handle unbalanced datasets by assigning higher class weight to the class with less samples [sk-unbalanced]. Naive Bayes is also able to handle unbalanced dataset by computing priors from the training data. However, the other techniques, such as decision trees, random forest and gradient boosted trees, do not handle unbalanced datasets on their own. To improve the performance of these other techniques, the training set of the larger classes should be divided into the same size as the smaller classes. Each base class can be trained on a portion of the larger classes and all of the smaller classer. The final classifier is then the average of all the base classifiers. In addition, methods can also be combined to handle unbalanced datasets. For instance, SVM classifier and Naive Bayes can be combined with random forest to yield better performance for classes where random forest is not able to classify.

The final insight, based on the feature extractions and results, is where improvements can be made to better classify audio heart sounds. In Table III, decision trees and random forest are not able to classify any extrasystole heartbeats. As well, based on the feature extraction results for dataset B, it is difficult to distinguish the features for extrasystole hearbeats. This suggests that better features should be extracted for dataset B so decision trees and random forest can find better splits for the data during training. Also from the results, the class precision are significantly better than the state of the art except for extrasystole and normal heartbeats, which are either slightly higher or lower. Based on this results, a more robust feature set that distinguishes normal heartbeats with the other classes should be extracted as well.

VII. Conclusion

In this paper, the process of classifying audio heart sounds is presented and machine learning techniques for this task are compared. Two audio heart sound datasets are used to train and verify the six chosen methods. The process to classify heartbeats includes preprocessing the datasets, extracting audio features, training methods and finally analyzing results. Preprocessing was done to normalize the data. Feature extraction then uses the preprocessed data to extract features using the whole signal and significant parts of the signal, such as S1 and S2. Using the feature extracted, six different classifiers are trained to classify the different heartbeats. Based on the individual class precisions, the results are significantly better than the state of the art results except for the precision for extrasystole and normal heartbeats. After comparing both the individual class precisions and F-scores of each dataset, the SVM classifier is found to be the most promising approach to classifying audio heart sounds. However, when selecting methods to use in practice, a classifier which provides explainability as well as high precision, such a gradient boosted trees, are preferred. Additionally, improvements to feature extraction techniques and handling of unbalanced datasets still need to be made to improve the result of these models.

References

[1] ”Diagnosis & Tests”, *WebMD*. [Online]. Available: https://www.webmd.com/heart-disease/guide/heart-disease-diagnosis-tests

[2] ”Heart Disease”, *Mayo Clinic*. [Online]. Available: https://www.mayoclinic.org/diseases-conditions/heart-disease/diagnosis-treatment/drc-20353124

[3] ”Heart Murmurs - Heart Sounds”, *Practical Clinical Skills*. [Online]. Available: https://www.practicalclinicalskills.com/heart-murmurs

[4] ”Medical Definition of Extrasystole”, *MedicineNet.com*. [Online]. Available: https://www.medicinenet.com/script/main/art.asp?articlekey=32159

[5] R. He et al., ”Classification of Heart Sound Signals Based on AR Model”, Computing in Cardiology Conference, 2016. [Online]. Available: https://pdfs.semanticscholar.org/41a6/4ff7803d769fbb0d0aefdd58f8835a-ec4ad1.pdf

[6] M. Singh and A. Cheema, ”Heart Sounds Classification using Feature Extraction of Phonocardiography Signal”, *International Journal of Computer Applications*, 2013. [Online]. Available: https://pdfs.semanticscholar.org/73a7/3eb140b1f7b496f801d697174392c172c55c.pdf

[7] H. Ryu, J. Park and H. Shin, ”Classification of heart sound recordings using convolution neural network”, *Computing in Cardiology*, 2016. [Online]. Available: https://ieeexplore.ieee.org/document/7868952/

[8] Y. Deng and P.J. Bentley, ”A Robust Heart Sound Segmentation and Classification Algorithm using Wavelet Decomposition and Spectrogram”, 2011. [Online]. Available: http://www.peterjbentley.com/heartworkshop/challengepaper3.pdf

[9] The PASCAL Classifying Heart Sounds Challenge 2011, Bentley P. and Nordehn G. and Coimbra M. and Mannor S. [Online]. Available: http://www.peterjbentley.com/heartchallenge/index.html

[10] T. Giannakopoulos, ”pyAudioAnalysis: An Open-Source Python Library for Audio Signal Analysis”, *PLOS*, 2015. [Online] Available: http://journals.plos.org/plosone/article/file?id=10.1371/journal.pone.0144610&type=printable

[11] D.S. Kim et al., ”Feature Extraction Based On Zero-Crossings With Peak Amplitudes For Robust Speech Recognition In Noisy Environments”, *International Conference on Acoustics, Speech and Signal Processing*,1996. [Online]. Available: https://www.researchgate.net/publication/3644271_Feature_Extraction_Based_On_Zero-Crossings_With_Peak_Amplitudes_For_Robust_Speech_Recognition_In_Noisy_Environments

[12] K. Chakraborty, A. Talele, and S. Upadhya, ”Voice Recognition Using MFCC Algorithm”, *International Journal of Innovative Research in Advanced Engineering*, 2014. [Online]. Available: http://ijirae.com/volumes/vol1/issue10/27.NVEC10086.pdf

[13] B.J. Mohan and R. Babu, ”Speech recognition using MFCC and DTW”, *Advances in Electrical Engineering*, 2014. [Online]. Available: https://ieeexplore.ieee.org/document/6838564/

[14] C. Ittichaichareon, S. Suksri and T. Yingthawornsuk, ”Speech Recognition using MFCC ”, *International Conference on Computer Graphics*, 2012. [Online]. Available: https://pdfs.semanticscholar.org/3439/454a00ef811b3a244f2b0ce770e80f7-bc3b6.pdf

[15] H. Zhang and D. Li, ”Naive Bayes Text Classifier”, *Granular Computing*, 2007. [Online]. Available: https://ieeexplore.ieee.org/document/4403192/

[16] Z. Fu et al., ”Learning Naive Bayes Classifiers for Music Classification and Retrieval”, *International Conference on Pattern Recognition*, 2010. [Online]. Available: https://ieeexplore.ieee.org/document/5597349/

[17] M.N. Murty and V.S. Devi, ”Bayes Classifier”, *Pattern Recognition*,pp 86-102, 2011. Available: https://link.springer.com/chapter/10.1007/978-0-85729-495-1_4

[18] A. Cheema and M. Singh, ”Heart Sounds Classification using Feature Extraction of Phonocardiography Signal”, *International Journal of Computer Applications*, 2013. Available: https://pdfs.semanticscholar.org/73a7/3eb140b1f7b496f801d697174392c172c55c.pdf

[19] ”Naive Bayes”, *Scikit Learn*. [Online]. Available: http://scikit-learn.org/stable/modules/naive_bayes.html

[20] ”Gaussian Naive Bayes”, *Scikit Learn*. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html

[21] S.Z. Li and G.D. Guo, ”Content-Based Audio Classification and Retrieval Using SVM Learning”, *IEEE Transactions on Neural Networks*, 2003. [Online]. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/content_audio_classification.pdf

[22] ”Support Vector Machines”,*Scikit Learn*. [Online]. Available: http://scikit-learn.org/stable/modules/svm.html

[23] ”RBF SVM parameters”, *Scikit Learn*. [Online]. Available: http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

[24] ”SVC”, *Scikit Learn*. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[25] ”Decision Trees”, *Scikit Learn*. [Online]. Available: http://scikit-learn.org/stable/modules/tree.html

[26] ”Decision Tree Classifier”, *Scikit Learn*. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[27] ”Decision Trees”, *Data Mining with Rattle and R*, pp 205-244, 2011. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4419-9890-3_11

[28] L. Breiman, ”Random Forests”, *Machine Learning*, 2001. [Online]. Available: https://link.springer.com/article/10.1023/A:1010933404324

[29] Felner JM, "The First Heart Sound", *Clinical Methods: The History, Physical, and Laboratory Examinations 3rd Edition,* 1990. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK333/

[30] Y. Freund and R.E. Schapire, ”A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting”, *Journal of Computer and System Sciences*, 1997. [Online]. Available: http://www.face-rec.org/algorithms/Boosting-Ensemble/decision-theoretic_generalization.pdf

[31] ”Ensemble Methods”, *Scikit Learn*. [Online]. Available: http://scikit-learn.org/stable/modules/ensemble.html

[32] ”AdaBoost Classifier”, *Scikit Learn*. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

[33] ”Random Forest Classifier”, *Scikit Learn*. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[34] ”Gradient Boosting Classifier”, *Scikit Learn*. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

[35] M. SokolovaNathalie and J. Szpakowicz, ”Beyond Accuracy, F-Score and ROC: A Family of Discriminant Measures for Performance Evaluation”, *Australasian Joint Conference on Artificial Intelligence*, 2006. [Online]. Available: https://pdfs.semanticscholar.org/7c6e/916c632b4c83777dd7184b923c97ecc-731ce.pdf

[36] ”SVM: Separating hyperplane for unbalanced classes”, *Scikit Learn*. [Online]. Available: http://scikit-learn.org/stable/auto_examples/svm/plot_separating_hyperplane_unbalanced.html

Appendix

A. Code Samples

Function 1. Preprocessing

```
import csv
import numpy as np
import wave
import struct
import pywt
def get_preprocessed_data(set_name, N=2,
    factor=10):
  """Get preprocessing data and label for a
      set"""
  """Inputs:
    set_name - either A or B
    N - minimum file sound length (seconds)
    factor - downsample factor"""
  filenames = []
  y_label = []
  framerates =[]

  if set_name.upper() == 'A':
    filenames, y_label, framerates =
        get_setA_file_label(N)
  else:
    filenames, y_label, framerates =
        get_setB_file_label(N)
  raw_data = get_raw_data(filenames) #get raw
      sound data
  x_data = np.array([down_sample(sample,
      factor=factor) for sample in raw_data])
      #downsample
  x_data = np.array([pywt.dwt(x,'db4')[0] for
      x in x_data]) #wavelet decomposition

  min_length = min(map(len, x_data))
  x_data = np.array([x[:min_length] for x in
      x_data]) #make all data the same length
  x_data = np.array([x/max(np.abs(x)) for x
      in x_data]) #normalize all data
  return x_data, y_label, framerates
```

Function 2. Finding peaks

```
from scipy.signal import find_peaks_cwt
import numpy as np

def find_peaks(samples, set_name):
  """Gets a list of peaks for each sample"""
  if set_name.upper() == 'A':
    interval = 200
    r = 5
  else:
    interval = 20
    r = 2
  all_peaks = []
  for sample in samples:
    indexes = find_peaks_cwt(sample,
        np.arange(1, r))
    peaks = []
    for i in indexes:
      if sample[i] > 0.15:
        peaks.append(i)

    if len(peaks) > 1:
      i = 1
      start = 0
      tmp_array = []
      max_peak = sample[peaks[start]]
      max_ind = start
      while i < len(peaks):
        if peaks[i] <= (peaks[start] +
            interval):
          if sample[peaks[i]] > max_peak:
            max_peak = sample[peaks[i]]
            max_ind = i
          if i == len(peaks)-1:
            tmp_array.append(peaks[max_ind])
            break
          i += 1
        else:
          tmp_array.append(peaks[max_ind])
          start = i
          max_ind = start
          max_peak = sample[peaks[start]]
          i += 1
      peaks = tmp_array
    all_peaks.append(peaks)
  return np.array(all_peaks)
```

Function 3. Finding S1 and S2 bounds

```
def get_S1S2_bounds(data, peaks, set_name):
    #finding difference between all peaks in
        every file
    all_diffs = []
    for k in range(len(peaks)):
        diff = np.diff(peaks[k])
        all_diffs.append(diff)

    #finding max difference or diastole period
    # and then labelling the first peak as s2
        and second peak as s1
    max_index = []
    s1s2_peaks = []
    for k in range(len(all_diffs)):
        if any(all_diffs[k]):
            max_index.append(np.argmax(
                all_diffs[k]))
            s2 = peaks[k][max_index[k]]
            s1 = peaks[k][max_index[k]+1]
            s1s2_peaks.append([s1, s2])
        else:
            max_index.append(-1)
            s1s2_peaks.append([-1,-1])
    s1s2_peaks = np.array(s1s2_peaks)

    #defining s1 and s2 boundaries
    s1_bounds = []
    s2_bounds = []
    if set_name == 'A':
        upper_s1 = 200*2
        lower_s1 = 80*2
        upper_s2 = 600*2
        lower_s2 = 70*2
    else:
        upper_s1 = 25*10
        lower_s1 = 10*10
        upper_s2 = 35*10
        lower_s2 = 10*10

    for k in range(len(s1s2_peaks)):
        if s1s2_peaks[k][0] == -1:
            s1_bounds.append([-1,-1])
            s2_bounds.append([-1,-1])
```

```
      else:
         s1_lower = s1s2_peaks[k][0]-lower_s1
         s1_upper = s1s2_peaks[k][0]+upper_s1
         s2_lower = s1s2_peaks[k][1]-lower_s2
         s2_upper = s1s2_peaks[k][1]+upper_s2
         if s1_lower < 0:
            s1_lower = 0
         if s2_lower < 0:
            s2_lower = 0
         if s1_upper >= len(data[0]):
            s1_upper = len(data[0]) - 1
         if s2_upper >= len(data[0]):
            s2_upper = len(data[0]) - 1
         s1_bounds.append([s1_lower,
             s1_upper])
         s2_bounds.append([s2_lower,
             s2_upper])

   return np.array(s1_bounds),
       np.array(s2_bounds)
```

Function 4. Getting standard deviation interval

```
def stdInterval(lower, low_index, upper,
    up_index, data):
   std = []
   for k in range(len(data)):
      if lower[k][0] == -1:
         std.append(0)
      else:
         dev = np.std(data[k]
            [lower[k][low_index]:
            upper[k][up_index]])
         if np.isnan(dev):
            std.append(0)
         else:
            std.append(dev)
   return np.array(std)
```

Function 5. Extracting features for set A (same steps for set B)

```
# Standard deviation for S1 and S2
s1_boundsA, s2_boundsA =
    get_S1S2_bounds(x_dataA, dataA_peaks, 'A')
stdS1_A =
    stdInterval(s1_boundsA,0,s1_boundsA,1,
    x_dataA)
stdS1_A = stdS1_A/max(stdS1_A)
stdS2_A =
    stdInterval(s2_boundsA,0,s2_boundsA,1,
    x_dataA)
stdS2_A = stdS2_A/max(stdS2_A)

# Mean and standard deviation for frequency
    spectrum of S1 and S2
freqS1_A = freqInterval(x_dataA,
    s1_boundsA,0,s1_boundsA,1)
stdS1_freqA = np.array([np.std(f) for f in
    freqS1_A])
stdS1_freqA = stdS1_freqA/max(stdS1_freqA)
meanS1_freqA = np.array([np.average(f) for f
    in freqS1_A])
meanS1_freqA = meanS1_freqA/max(meanS1_freqA)
freqS2_A = freqInterval(x_dataA,
    s2_boundsA,0,s2_boundsA,1)
stdS2_freqA = np.array([np.std(f) for f in
    freqS2_A])
stdS2_freqA = stdS2_freqA/max(stdS2_freqA)
meanS2_freqA = np.array([np.average(f) for f
    in freqS2_A])
meanS2_freqA = meanS2_freqA/max(meanS2_freqA)

# zero crossing rate of frame
zero_crossingsA =
    np.array([audioFeatureExtraction.stZCR(x)
    for x in x_dataA])
zero_crossingsA =
    zero_crossingsA/max(zero_crossingsA)

# signal energy of frame
energyA = np.array([audioFeatureExtraction.
    stEnergy(x) for x in x_dataA])
energyA = energyA/max(energyA)

# entropy of energy
entropyA = np.array([audioFeatureExtraction.
    stEnergyEntropy(x, numOfShortBlocks=50)
    for x in x_dataA])
entropyA = entropyA/max(entropyA)

# frequency domain
X_dataA = np.array([np.fft.fft(x) for x in
    x_dataA])
X_dataA = np.array([np.abs(X)/max(np.abs(X))
    for X in X_dataA])

# spectral entropy
entropy_freqA =
    np.array([audioFeatureExtraction.
    stSpectralEntropy(X,
    numOfShortBlocks=150) for X in X_dataA])
entropy_freqA =
    entropy_freqA/max(entropy_freqA)

# spectral flux
fluxA =
    np.array([np.abs(audioFeatureExtraction.
    stSpectralFlux(X[:int(len(X)/2)],
    X[int(len(X)/2):])) for X in X_dataA])
fluxA = fluxA/max(fluxA)

# spectral centroid of frame (given abs(FFT))
FsA = int(framerate_A[0]/20) # framerate is
    the same for all of set A
centroidA = np.array([audioFeatureExtraction.
    stSpectralCentroidAndSpread(np.abs(X),FsA)
    for X in X_dataA])
centroidA[:,0] =
    centroidA[:,0]/max(centroidA[:,0])
centroidA[:,1] =
    centroidA[:,1]/max(centroidA[:,1])

from python_speech_features import mfcc

mfccA =
    np.array([mfcc(np.abs(x),samplerate=FsA,
    numcep=3,winlen=0.025) for x in x_dataA])
mfccA_feat = np.array([[np.average(m[:,0]),
    np.std(m[:,0]), np.average(m[:,1]),
    np.std(m[:,1]), np.average(m[:,2]),
    np.std(m[:,2])] for m in mfccA])
for i in range(len(mfccA_feat[0])):
   mfccA_feat[:,i] = np.abs(mfccA_feat[:,i])/
```

```
        max(np.abs(mfccA_feat[:,i]))
```

Function 6. Pre-Training steps

```
x_utrainA = np.column_stack((zero_crossingsA,
    energyA, entropyA, entropy_freqA, fluxA,
    centroidA[:,1], mfccA_feat, stdS1_A,
    stdS2_A, stdS1_freqA, meanS1_freqA,
    stdS2_freqA, meanS2_freqA))
x_utrainB = np.column_stack((zero_crossingsB,
    energyB, entropyB, entropy_freqB, fluxB,
    centroidB[:,1],mfccB_feat,stdS1_B,
    stdS2_B, stdS1_freqB, meanS1_freqB,
    stdS2_freqB, meanS2_freqB))

from sklearn import preprocessing
le = preprocessing.LabelEncoder()
y_utrainA = le.fit_transform(y_labelA)
y_utrainB = le.fit_transform(y_labelB)

from sklearn.utils import shuffle
x_trainA, y_trainA = shuffle(x_utrainA,
    y_utrainA)
x_trainB, y_trainB = shuffle(x_utrainB,
    y_utrainB)

from sklearn.model_selection import KFold
kf = KFold(n_splits=4, shuffle=False)
```

Function 7. Naive Bayes classifier training and validation for set A (same steps for set B and other classifiers)

```
from sklearn.naive_bayes import GaussianNB
resultsA_gnb = []
gnbA = [] # the models are saved here
i = 0
precisionsA = {0: [], 1:[], 2:[], 3:[]}
p_GNBA = []
r_GNBA = []

for train_index, test_index in
    kf.split(x_trainA):
   cm = np.zeros((4,4))
   TP_A = {0:0, 1:0, 2:0, 3:0}
   FP_A = {0:0, 1:0, 2:0, 3:0}
   gnbA.append(GaussianNB(priorsA))
   resultsA_gnb.append(gnbA[i].fit(
       x_trainA[train_index],
       y_trainA[train_index]).
       score(x_trainA[test_index],
       y_trainA[test_index]))
   preds =
       gnbA[i].predict(x_trainA[test_index])
   actual = y_trainA[test_index]
   print('Predic ',i, ': ', preds)
   print('Actual ',i, ': ', actual)
   for p in range(len(preds)):
      cm[actual[p]][preds[p]] += 1
      if preds[p] == actual[p]:
         TP_A[preds[p]] += 1
      else:
         FP_A[preds[p]] += 1
   print(TP_A)
   print(FP_A)
   print(cm)
   for n in range(4):
      if TP_A[n] == 0 and FP_A[n] == 0 :
         precisionsA[n].append(0)
      else:
         precisionsA[n].append(float(TP_A[n]/
             (TP_A[n]+FP_A[n])))
   TP = cm[0][0] + cm[1][1] + cm[2][2] +
       cm[3][3]
   FP = np.sum(cm[1:,0]) +
       np.sum(cm[0:1,1])+np.sum(cm[2:,1]) +
       np.sum(cm[0:2,2]+cm[3:,2]) +
       np.sum(cm[0:3,3])
   FN = np.sum(cm[0,1:]) +
       np.sum(cm[1,0:1])+np.sum(cm[1,2:]) +
       np.sum(cm[2,0:2])+np.sum(cm[2,3:]) +
       np.sum(cm[3,0:3])
   p_GNBA.append(float(TP/(TP+FP)))
   r_GNBA.append(float(TP/(TP+FN)))
   i += 1
resultsA_gnb = np.array(resultsA_gnb)
print('Overall Average Precision: ',
    np.average(p_GNBA), ' breakdown: ',
    p_GNBA)
print('Overall Average Recall: ',
    np.average(r_GNBA), ' breakdown: ',
    r_GNBA)
print('Precisions: ', precisionsA)
print('Average Precisions for each class: ',
    np.average(precisionsA[0]),
    np.average(precisionsA[1]),
    np.average(precisionsA[2]),
    np.average(precisionsA[3]))
print('Accuracy of splits: ', resultsA_gnb)
print('Average results: ',
    np.average(resultsA_gnb))
```