Brigham Young University
BYU ScholarsArchive

Theses and Dissertations

2018-10-04

Using Machine Learning to Accurately Predict Ambient Soundscapes from Limited Data Sets

Katrina Lynn Pedersen
Brigham Young University

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Physical Sciences and Mathematics Commons

BYU ScholarsArchive Citation
Pedersen, Katrina Lynn, "Using Machine Learning to Accurately Predict Ambient Soundscapes from Limited Data Sets" (2018). *Theses and Dissertations*. 9272.
https://scholarsarchive.byu.edu/etd/9272

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact ellen_amatangelo@byu.edu.

المنارة للاستشارات

www.manaraa.com

Using Machine Learning to Accurately Predict Ambient Soundscapes from Limited Data Sets

Katrina Lynn Pedersen

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Mark Transtrum, Chair
Kent Gee
Sean Warnick

Department of Physics and Astronomy

Brigham Young University

Copyright © 2018 Katrina Lynn Pedersen

All Rights Reserved

المنارة للاستشارات

www.manaraa.com

ABSTRACT

Using Machine Learning to Accurately Predict Ambient
Soundscapes from Limited Data Sets

Katrina Lynn Pedersen
Department of Physics and Astronomy, BYU
Master of Science

The ability to accurately characterize the soundscape, or combination of sounds, of diverse geographic areas has many practical implications. Interested parties include the United States military and the National Park Service, but applications also exist in areas such as public health, ecology, community and social justice noise analyses, and real estate. I use an ensemble of machine learning models to predict ambient sound levels throughout the contiguous United States. Our data set consists of 607 training sites, where various acoustic metrics, such as overall daytime L_{50} levels and one-third octave frequency band levels, have been obtained. I have data for 117 geospatial features for the entire contiguous United States, which include metrics such as distance to the nearest road or airport, and the percentage of industrialization or forest in a specific area. I discuss initial model predictions in the spatial, frequency, and temporal domains, and the statistical advantages of using an ensemble of machine learning models, particularly for limited data sets. I comment on uncertainty quantification for machine learning models originating from limited data sets.

Keywords: acoustics, ensemble model, machine learning, soundscape, statistics, uncertainty quantification

المنارة للاستشارات

www.manaraa.com

ACKNOWLEDGMENTS

Thanks to all of my family who have supported and encouraged me and made it possible for me to get an education.

I would also like to thank Dr. Mark Transtrum for his patience and guidance as we worked on this problem. I am grateful to Brigham Young University for giving me an opportunity to study and gain an education. Thanks also to the rest of my committee, Dr. Kent Gee and Dr. Sean Warnick, and Brooks Butler for their insightful comments and questions.

Thanks to Blue Ridge Reasearch and Consulting, LLC (BRRC) for obtaining and organizing a database of geospatial features and acoustic metrics. Additionally, thanks to BRRC for providing the code to create maps of sound level predictions.

This work was supported by BRRC via a U.S. Army Small Business Innovation Research (SBIR) contract.

المنارة للاستشارات

www.manaraa.com

Contents

Table of Contents iv

List of Figures vi

1 Introduction 1
1.1 Motivation 1
1.1.1 Geospatial Acoustics 1
1.1.2 Limited Data Sets in Machine Learning 3
1.2 Background 4
1.2.1 Acoustics and Sound Level Metrics 4
1.2.2 Principles of Machine Learning 8
1.2.3 Uncertainty Quantification 18
1.3 Previous Research 20
1.3.1 Literature Review 20
1.4 Objectives 22

2 Methodology 24
2.1 Data Sets 24
2.2 Computational Pipeline 28
2.3 Model Selection and Parameter Tuning 29
2.4 Feature Reduction 29

3 Results 32
3.1 Ensemble of Models 32
3.1.1 Model Selection 32
3.1.2 Ensemble Advantages 34
3.2 Maps 39
3.2.1 Summer L_{10}, L_{50}, and L_{90} Day and Nighttime Maps 39
3.2.2 Frequency Group Maps 39
3.2.3 Hourly Summer L_{50} Frequency Group Maps 42
3.3 Leave-Four-Out Validation Study 44
3.4 Feature Reduction 54

المنارة للاستشارات
www.manaraa.com

3.4.1 Initial Feature Importance Rankings 54
3.4.2 Error of Reduced Feature Model 56
3.4.3 Changes in Predictions from Reduced Feature Models 56

4 Conclusions and Future Work **62**
4.1 Conclusion 62
4.2 Future Work 62

Appendix A Geospatial Features **64**

Appendix B Acoustic Data **68**

Appendix C Pipeline Code **70**
C.1 README 70
C.2 Main.py 71
C.3 ParseData.py 72
C.4 PipelinesPool.py 78
C.5 PrunedModels.py 87
C.6 Scalers.py 89
C.7 Validation.py 98
C.8 AnalysisTools.py 100
C.9 createFullConus.py 103
C.10 findMedian.py 103
C.11 Sample Data File: Data1008.py 104

Bibliography **105**

المنارة للاستشارات
www.manaraa.com

List of Figures

1.1 Statistical Sound Pressure Levels 5
1.2 A-Weighting Curve 7
1.3 Random Forest Graphic 10
1.4 Neural Network Graphic 12
1.5 K-Nearest Neighbors Graphic 13
1.6 Support Vector Machine Graphic 15
1.7 Gaussian Process Regression Graphic 17

2.1 Mean Upward Radiance at Night 26
2.2 Proportion of Forest Landcover 27
2.3 Computational Pipeline 29

3.1 Residuals from Leave-One-Out Cross Validation 33
3.2 Gradient Boosted Regressor Predictions 35
3.3 Neural Network Predictions 36
3.4 Ensemble Model Predictions 37
3.5 Standard Deviation Map 38
3.6 Day and Night Predicted Exceedance Levels 40
3.7 Ensemble Standard Deviation of Day and Night Predicted Exceedance Levels . . . 41

المنارة للاستشارات

www.manaraa.com

3.8 Ensemble Daytime L_{50} Frequency Groups in North Carolina 43
3.9 Ensemble High and Low Frequency Group Predictions for Asheville (4 a.m.) . . . 44
3.10 Ensemble High and Low Frequency Group Predictions for Asheville (8 a.m.) . . . 45
3.11 Ensemble High and Low Frequency Group Predictions for Asheville (10 p.m.) . . . 46
3.12 Leave-One-Out Daytime Predictions at First Two Validation Sites 47
3.13 Leave-One-Out Daytime Predictions at Last Two Validation Sites 48
3.14 Leave-One-Out Nighttime Predictions at First Two Validation Sites 49
3.15 Leave-One-Out Nighttime Predictions at Last Two Validation Sites 50
3.16 Day and Nighttime Leave-Four-Out Predictions at First Two Validation Sites 52
3.17 Day and Nighttime Leave-Four-Out Predictions at Last Two Validation Sites 53
3.18 Error as a Function of the Number of Features (Gini Importance) 57
3.19 Error as a Function of the Number of Features (Neural Network Importance) 58
3.20 Change in Daytime L_{50} Predictions with Reduced Features (Gini Importance) . . . 59
3.21 Change in Daytime L_{50} Predictions with Reduced Features (Human Intuition) . . . 60

المنارة للاستشارات
www.manaraa.com

Chapter 1

Introduction

In this thesis, I examine two problems: the accurate prediction of geospatial sound levels, and the practice of creating, validating, and improving machine learning models with limited data. This chapter provides motivation for both problems, and also provides a summary of various acoustic metrics and machine learning principles. Additionally, an overview of uncertainty quantification and previous research is given.

1.1 Motivation

1.1.1 Geospatial Acoustics

The ability to accurately characterize the soundscape, or combination of sounds, of various geographic areas has broad applications. It is specifically valuable to the United States military and the National Park Service (NPS). It also holds weight in areas such as epidemiology and ecology.

In the military, soundscape characterization is important to both avoid aural detection and improve current detection of foreign aircraft. If we understand the limits of enemy detection,

المنارة للاستشارات
www.manaraa.com

we will be able to get closer to foreign lands while avoiding detection. Therefore, knowledge of the soundscape in an area may aid mission planning.

The NPS created the Natural Sounds and Night Skies Division to protect and restore the natural soundscapes of the national parks [1, 2]. Natural sounds, or the lack thereof, affect the experience of visitors to the national parks. In particular, recreational motorized noise has been shown to negatively impact visitor appreciation for natural landscapes [3]. Research suggests natural soundscapes play an important role in visitor experiences and ecological community processes within national parks [2,4]. A study of a national park in Spain found that visitors were annoyed by various anthropogenic noises while visiting the park, and were willing to pay a small entrance fee to support a noise-reduction program [5]. Experiencing natural sounds is an important motivation for visiting natural areas, such as the national parks [4].

Commercially, there are possible applications in community and social justice noise analyses, real estate, epidemiological community health studies, and ecology. Predicting soundscapes may aid in making decisions regarding construction in or near inhabited areas. Ambient sound levels also affect housing prices and are correlated with depression and anxiety [6], as well as hypertension [7–9]. In addition, epidemiological studies have found that increased noise may be associated with changes in blood pressure, heart rate, and stress [7,9]. More specifically, a positive correlation has been made between aircraft noise and cardiovascular risk [7, 9] and impaired reading comprehension, recognition memory, and motivation in children [10]. In adults, sound that varies significantly in pitch, timbre, or tempo over time has been shown to impair cognitive function as well [11]. As the number and quality of epidemiological studies continues to increase [9], providing access to a complete characterization of soundscapes over various geographical areas will aid in identifying correlations between health and ambient sound levels.

Noise has also been linked to altering behavior, communication, and physiological state among several animals, such as birds [12–15], marine life [16–20], and frogs and toads [21]. Changes in

المنارة للاستشارات
www.manaraa.com

a soundscape have the ability to affect any animal however [14]. High levels of ambient noise can lead to masking of communication within a species and also masking of predator noise, particularly if the ambient noise is at the same frequency at which a species usually communicates [14, 15, 17, 18, 20]. This leads species to occasionally alter the time of day they are active, or to alter the frequency with which they communicate [14, 15, 17, 18, 20]. Some species may also attempt to increase the amplitude of their acoustic communication [14, 17]. The acoustic complexity of a region (not including anthropogenic noise) has also been seen to correlate with higher biodiversity [19, 22]. Accurate characterization of soundscapes will help ecologists determine the effects of a variety of ambient sound levels on ecosystem dynamics.

1.1.2 Limited Data Sets in Machine Learning

Machine learning is comprised of a group of modeling techniques that use existing data to "learn" some function relating inputs to outputs. After learning, the model is used to make predictions on novel instances. More details on machine learning are given in Sec. 1.2.2. Machine learning is generally performed on large data sets that are statistically similar to the data set on which predictions will be made. Machine learning may also be used on limited data sets, but the success of machine learning models is often dependent upon the ability of the data to characterize all aspects of model behavior. Transfer learning has been applied successfully to limited data sets when large amounts of labeled data are available for a task similar to the desired task [23].

Additionally, validation methods that are commonly used on big data sets are not well suited to machine learning with limited data sets. Holdout validation methods generally require a large amount of data to be removed during the learning, or training, process to use as a test set after training. When large data sets are used, training data are not significantly affected by removing a test set, so validation measures are likely a good approximation of the performance of the complete model. However, holding out data for testing from a limited data set may significantly alter the

المنارة للاستشارات
www.manaraa.com

trained model, assuming most instances carry unique information. In other words, in limited data sets, each instance generally has a greater effect on the learning process than in big data sets. This suggests that models trained on limited data sets are more likely to be sensitive to the removal of data for validation. No standard practice currently exists for measuring performance error of machine learning models trained on limited data sets.

Two important problems to consider are how to best improve machine learning models in the limited data regime when transfer learning is not possible, and how to best measure the performance of these models.

1.2 Background

1.2.1 Acoustics and Sound Level Metrics

Acoustic metrics are used to summarize noise across given temporal or frequency ranges. The sound pressure level (SPL), or level, is defined in terms of the root mean square pressure:

$$L_p = SPL = 20\log_{10}(\frac{P_{rms}}{P_{ref}})$$

where P_{ref} is the reference pressure of the surrounding fluid and the SPL has units of decibels (dB). In air, P_{ref} is close to 20 μPa. Note that SPL is measured on a logarithmic scale, so a doubling of P_{rms} corresponds to a 6 dB increase in level. To give some examples of approximate noise levels, rustling leaves are 20 dB, whispering is 30 dB, conversational speech is 60 dB, and jet noise is 150 dB.

Statistical noise levels may be calculated from SPLs over any given time domain. More specifically, the n-percent exceeded level, L_n, is the level exceeded n-percent of the time. The L_{10}, L_{50}, and L_{90} are shown for an acoustic signal in Figure 1.1. Since the L_{10} is the noise level exceeded 10 percent of the time, it will always be higher than the L_{50}, assuming there is variation

المنارة للاستشارات
www.manaraa.com

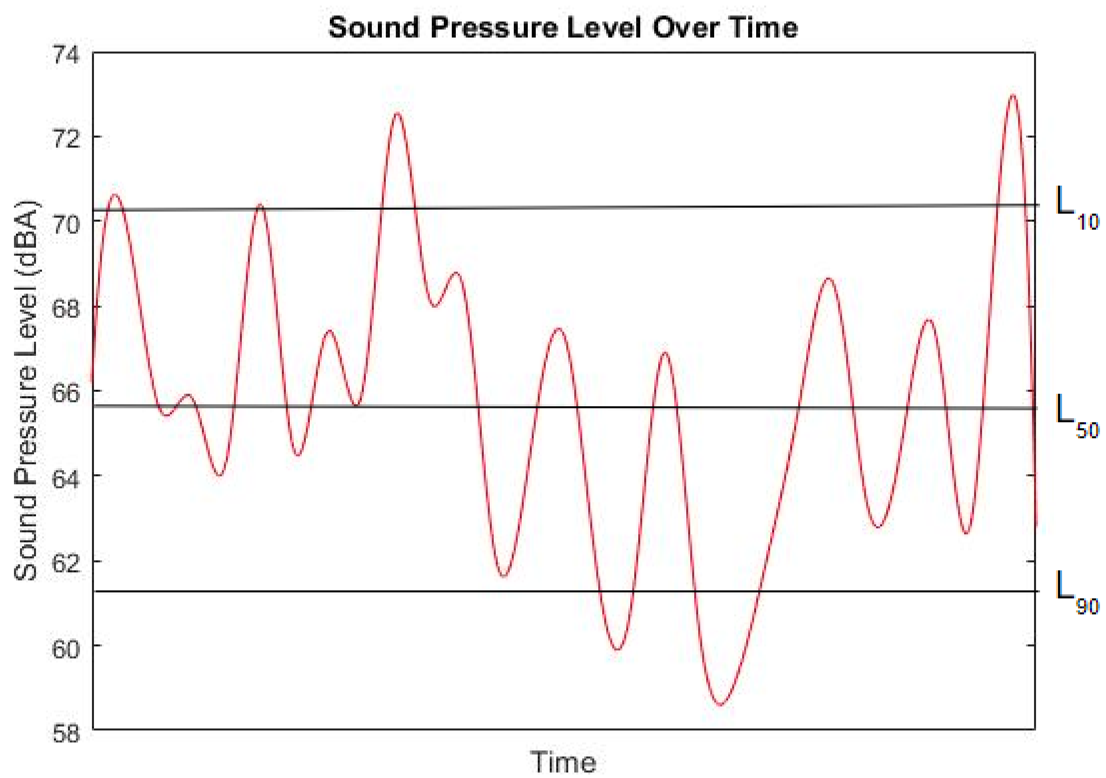

Figure 1.1 Plot of the SPL as a function of time. The L_{10}, L_{50}, and L_{90} are labeled.

www.manaraa.com

in the acoustic signal. Similarly, the L_{50} will always be higher than the L_{90}. The L_{10} represents the upper limits of fluctuation of the acoustic signal due to atypical or sporadic events, such as traffic noise. The L_{90}, on the other hand, is characteristic of the background noise level.

In addition to n-percent exceeded levels, the equivalent continuous sound level, or L_{eq}, can be used to describe the average sound energy over a given time. It is calculated from the mean square pressure over some total time T from T_1 to T_2 as:

$$L_{eq} = 10\log_{10}\left(\frac{1}{P_{ref}^2}\frac{1}{T}\int_{T_1}^{T_2} P^2(t)dt\right).$$

Common time intervals are hourly, daytime (7 a.m.-7 p.m.), nighttime (7 p.m.-7 a.m.), and seasonal.

The frequency domain of an acoustic signal may also be specified. This is commonly done for fractional octave spectra, such as one-third octave bands. Constant bandwidth spectra, in which the acoustic source is measured using equally spaced frequency bands, are not as useful as fractional octave spectra due to the frequency response of the human ear. One-third octave spectra are determined such that the upper (f_u) and lower (f_l) band edges have a ratio $f_u/f_l = 2^{1/3}$. The center frequency (f_c) of a band is determined by the geometric mean: $f_c = \sqrt{f_l f_u}$. Standard one-third octave bands were used in the measurement process.

In addition to one-third octave bands, three groups of bands were used as part of this thesis to analyze low (12.5-125 Hz), medium (160-1,250 Hz), and high (1,600-12,500 Hz) frequency noise. The SPL for a given group (SPL_g) was calculated from the k one-third octave spectral levels within the given group:

$$SPL_g = 10\log_{10}\left(\sum_k 10^{L_k/10}\right).$$

In addition to the decibel scale, other scales have been created to weight sound according to how the human ear reacts to different frequencies. Flat-weighted or unweighted levels often use units of dBZ since there is zero frequency weighting. The only other weighting discussed here is

المنارة للاستشارات
www.manaraa.com

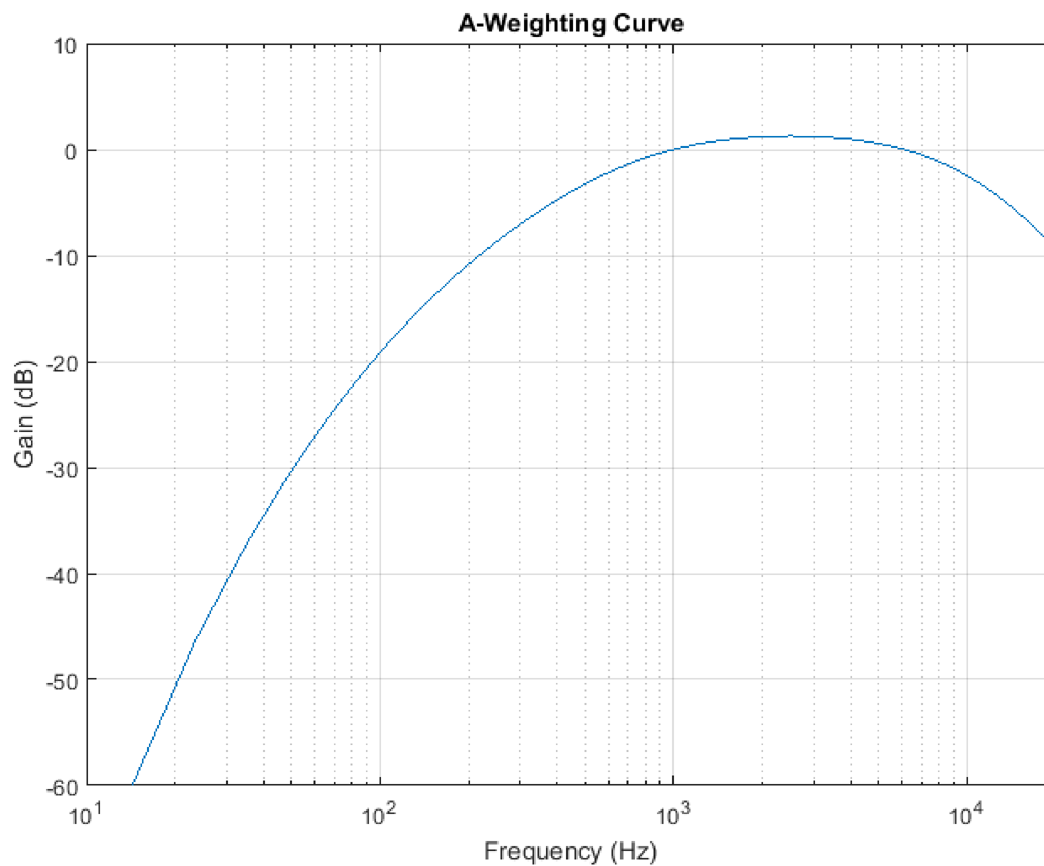

Figure 1.2 Plot of the A-weighting for frequencies from 10 Hz to 20 kHZ. The A-weighting is added to flat-weighted frequency dependent SPLs to calculated the A-weighted SPLs.

the A-weighting, which uses units of dBA. The A-weighting was developed to impose a similar frequency response as that of the human ear. Sound in the 1-6 kHz range is increased slightly, while levels are decreased outside of that range. To calculate A-weighted levels, the A-weighting curve (see Figure 1.2) is added to flat-weighted levels.

In this thesis, flat-weighted levels are used for all frequency measurements. However, all other metrics will use A-weighted values, which are fairly standard for n-percent exceeded and equivalent continuous sound levels.

المنارة للاستشارات
www.manaraa.com

1.2.2 Principles of Machine Learning

Supervised and Unsupervised Learning

There are two main types of machine learning, supervised and unsupervised. Supervised machine learning models fit model parameters to accurately map from an input space to an output space, and are used to make predictions in the output space for novel input data. Input data are comprised of a set of features. Supervised methods require a labeled training data set in which input and output values are provided for all training instances. Prior to the training, or learning, process, model hyperparameters, such as the maximum number of iterations or some error tolerance, are specified by the user. During the training process, parameters are fit in attempts to minimize the loss function of the model. After a supervised model has been trained, new instances, consisting of novel input data, may be given to the model. The model provides predicted output values based on the values of its parameters and hyperparameters.

On the other hand, unsupervised methods do not make predictions, but are used to look for patterns or unique structures in a single data set. I will focus on applications of supervised machine learning in this thesis since I want to predict ambient sound levels. Nevertheless, there are applications of unsupervised methods that can aid in better understanding the data set and identifying potential areas to improve it. As mentioned before, there is no standard method for improving and effectively measuring the error of supervised machine learning models created from limited data sets. However, unsupervised learning methods can help identify instances that are underrepresented in the training data set.

In the following subsections, various supervised machine learning models are described. The descriptions are not comprehensive, but rather a summary of the algorithms. For further information, the reader is directed to Marsland's text, *Machine Learning: An Algorithmic Perspective* [24], or Bishop's text, *Pattern Recognition and Machine Learning* [25].

Most supervised models are able to handle both classification and regression problems with

المنارة للاستشارات
www.manaraa.com

minimal changes in the algorithm. Although I will focus on the regression problem when predicting acoustic levels, several of the descriptions provided below address the classification problem, which is often easier to visualize.

Gradient Boosted Regression Trees

Gradient boosted regression trees (GBR) utilize an ensemble of decision trees and gradient boosting. The random forest algorithm is similar to GBR because both are composed of an ensemble of decision trees. For simplicity, we first look at a graphic of a trained random forest (see Figure 1.3) and examine how predictions are made after training. Each of the three shown decision trees is comprised of several nodes, or junctions. At each junction, some Boolean question is asked, and the resulting answer determines the path that is followed along each tree. The figure shows three specific trees of n total trees, which make up the forest. When using the forest to make predictions, we first find the final outcome or prediction of each individual tree. The most popular outcome from all n trees determines the predicted output of the forest for a given instance. In the regression model, each tree predicts a continuous value, rather than a classification.

There are various algorithms for creating decision trees, but I will use the ID3 algorithm [26] to illustrate the training process of a single tree here. The ends of a tree where predictions are made are called leaves, and the lines connecting nodes are branches. If all training instances have the same value, a leaf is returned with that value. If there are no features left, a leaf is returned with the most common or average value. Otherwise, a node is created that splits according to the feature that maximizes the information gain, and then that feature is removed from the data set. Branches are added from that node for all remaining features and the information gain of each feature is recalculated. This process is iterated through until a tree is complete.

Boosting is the process of taking an ensemble of weak learners, which perform slightly better than chance, and using them to make fairly good predictions. Assuming most of the trees give

المنارة للاستشارات
www.manaraa.com

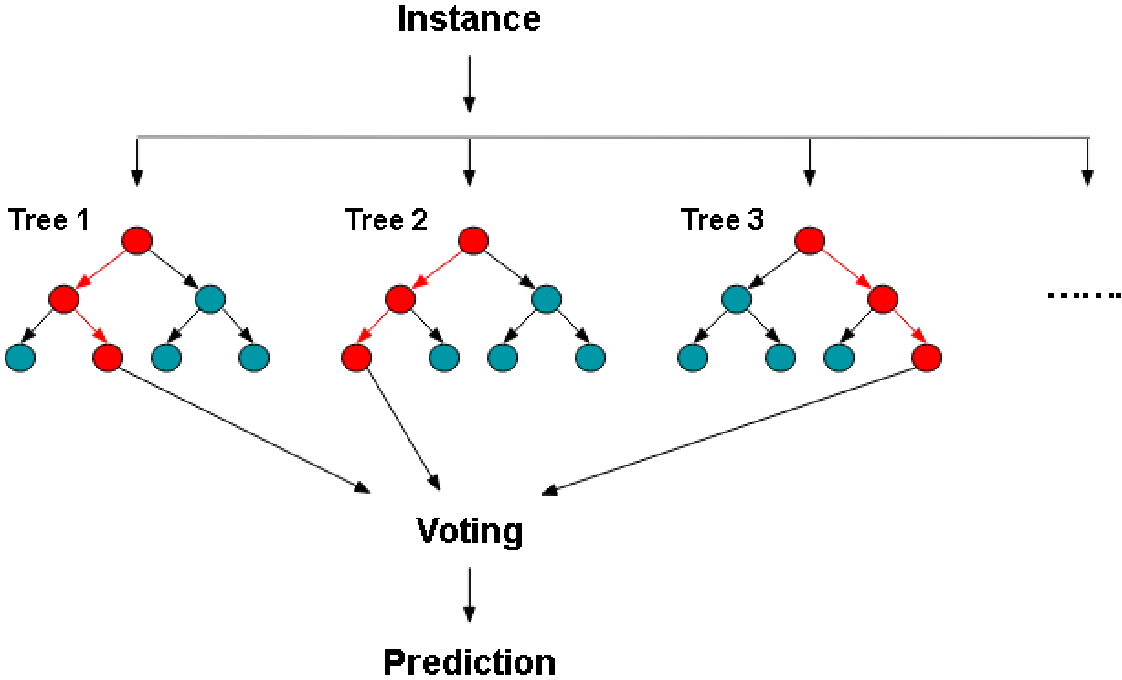

Figure 1.3 A simplified graphic of a RF model built from classification trees.

المنارة للاستشارات
www.manaraa.com

the correct prediction for most of the data, and assuming that most incorrect predictions will not give the same incorrect prediction, a boosted forest should make an accurate prediction. One advantage of using GBR is that decision trees produce inspectable models [27]. In other words, it is possible to identify which features are higher up on the decision trees, and therefore more important to model predictions. This can be advantageous when trying to understand model behavior. Additionally, GBR are better than a single decision tree at avoiding overfitting because they use a boosted ensemble.

Neural Networks

Artificial neural networks (NNs) were inspired by the process of learning in the brain. Figure 1.4 shows a simplified graphic of a NN, where each node is loosely based off of a neuron. Although the graphic only shows two hidden layers of nodes, it is possible to have many hidden layers between the input and output nodes. It is also possible to only have one or zero hidden layers. Generally, NNs with more hidden layers are better at modeling systems with high levels of complexity, but they are also more prone to overfitting. All connections between nodes weight the output from the previous node to the next node as information propagates from the left to the right of the graphic. Each node may perform a different function, call an activation function, on the weighted input that it receives. The activation function determines the output of a given node. The simplest nontrivial activation function is a step function. If the input to a given node is above some threshold, it will fire and send on the value 1. Otherwise, it will send on 0. NNs train by first stepping forward through the model with a given instance, and then propagating the error back through the model to adjust weights.

One type of NN is a multilayer perceptron. During the training phase for a multilayer perceptron, all weights for the hidden and output nodes are first initialized to small random numbers. The first instance is fed into the model and the activation of each neuron is calculated

المنارة للاستشارات
www.manaraa.com

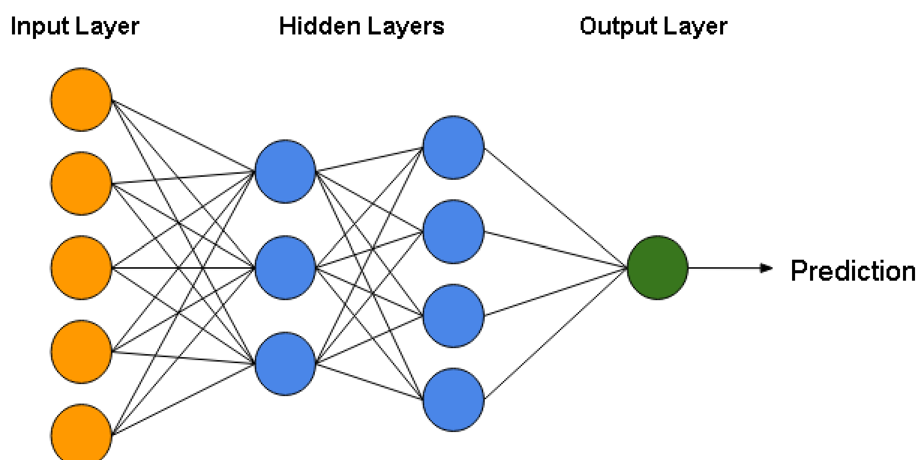

Figure 1.4 A simplified graphic of a NN with two hidden layers.

until the output node is reached. The error for the output node is calculated, and then the error in the hidden layers is found. The output and hidden layer weights are updated according to their error and the learning rate, a hyperparameter which determines how rapidly the weights change. The process is repeated until learning stops and error is stable. After the model is trained and all weights have been tuned, predictions are made by feeding instances through the model.

Unlike GBR, NNs with many hidden layers are difficult to inspect for physical meaning. However, the relationships between input features and model outputs become more inspectable when fewer hidden layers are used.

K-Nearest Neighbors

The k-nearest neighbors (KNN) algorithm is different than GBR and NNs in that no training is required before predictions are made. The algorithm works by locating the k-nearest instances from the training data set in the input data space, and then taking the average of the corresponding k-outputs. It is possible to use different distance metrics. For example, the distances to all k-outputs are sometimes weighted according to their distance from the new training instance. It is important to normalize all features prior to calculating the k-nearest neighbors in order to give

المنارة للاستشارات
www.manaraa.com

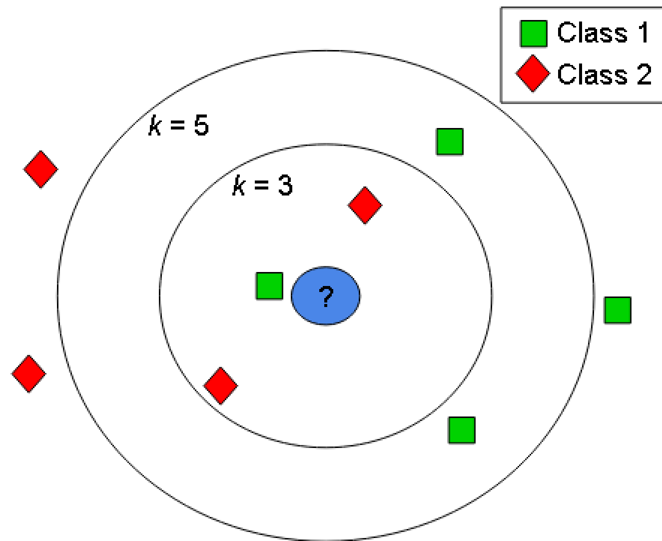

Figure 1.5 A graphic showing a KNN classification algorithm. If $k = 3$, the new example will be classified as a red diamond, but if $k = 5$, the new example will be classified as a green square.

equal weight to distance contributions from each feature.

Figure 1.5 shows a KNN classification model for k equal to 3 and 5 for a novel instance. If $k = 3$, the classification would be a red diamond, but if $k = 5$, the classification would be a green square. This algorithm can be computationally expensive for models with large training sets and numbers of input features because all distances must be calculated before identifying the k-nearest instances. There are a few tricks to increase the efficiency of the algorithm, but KNN is generally slow for large data sets.

Support Vector Machines

Before explaining how support vector machines (SVMs) are trained and make predictions, it is important to understand the kernel trick, or kernel substitution. The kernel trick was inspired by a desire to linearly separate data that was not linearly separable in the initial feature space. If the data could be transformed into higher dimensional spaces, it is possible that it would become linearly separable. For problems which only rely on the inner product of input vectors, the original

المنارة للاستشارات
www.manaraa.com

inner product may be replaced by an inner product with a different choice of kernel. Consider the nonlinear mapping $\phi(x)$ corresponding to a kernel function $k(x,x') = \phi(x)^T\phi(x')$. It is often possible to calculate $k(x,x')$ without knowing $\phi(x)$ or $\phi(x')$ because $k(x,x')$ can be expressed in terms of the inner product of x and x'. The kernel is a symmetric function that allows us to work in a high-dimensional space without ever truly transforming all data into that space or performing calculations there. Kernel trick methods are commonly used for nearest neighbor methods, such as KNN. Additionally, they are used for models, such as SVMs, that aim to partition data according to the distance between different training points.

SVMs attempt to separate data such that data with similar output values are grouped together. Decision boundaries, or hyperplanes, are used to partition data by creating a maximum margin between different clusters as seen in Figure 1.6. Instances that are on the margin (the blue triangle outline and red square outline) are called support vectors and are found during the training process as a chosen loss function, such as the least-squares error, is minimized. SVMs handle continuous data by using a free parameter ε to determine the maximum tolerable difference between predictions of members in the same grouping. This helps limit the number of support vectors. When making predictions, data that falls within ε of a boundary is predicted to have a similar output value as other members within that boundary.

Kernel Ridge Regression

Kernel ridge (KR) regression uses the kernel trick and then linear regression to fit the data. A KR model is very similar to a SVM, but instead of using some free parameter ε to determine the spread allowed near a decision boundary, KR uses squared error loss.

المنارة للاستشارات
www.manaraa.com

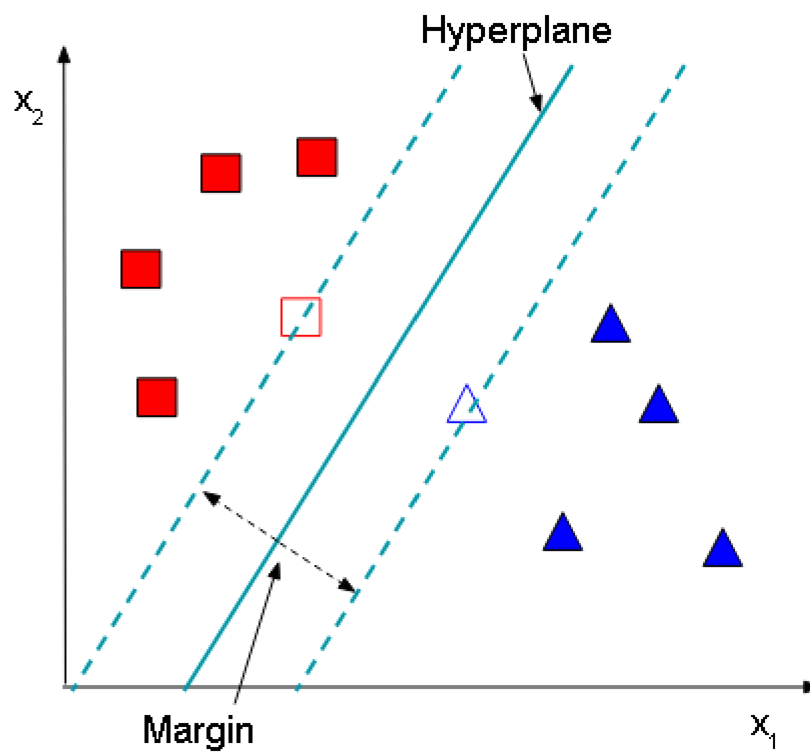

Figure 1.6 A graphic showing how SVMs partition data. The axes are input parameters and the squares and triangles represent different classes or output values.

المنارة للاستشارات
www.manaraa.com

Gaussian Process Regression

Gaussian process regression (GPR) is a stochastic process, or collection of random variables, that uses functions sampled from a multivariate Gaussian distribution to identify probable predicted values for novel instances. Figure 1.7 shows the prior and posterior distributions and predictions of a Gaussian process. Prior distributions are generated as functions sampled from a multivariate Gaussian distribution, generally with a mean of zero and covariance specified by the available input data. In order to obtain the covariance matrix of the input data, it is necessary to choose a kernel, such as the squared exponential or linear kernel. After priors have been generated, the training data are used to constrain the functions and create the posterior distributions for all functions as seen in the middle panel of Figure 1.7. Since all functions are Gaussian it is possible to calculate confidence intervals of the predicted data. The plot on the right of Figure 1.7 shows the mean prediction and one standard deviation above and below the mean. One of the advantages of GPR is the availability of confidence intervals. Locations that have large confidence intervals identify areas where the model could most benefit from the addition of new training data. GPR is not as efficient in high dimensional spaces, particularly for more complex kernel functions, so the training process often requires more time for data sets with large numbers of features.

Validation Methods

When large amounts of labeled data are available, it is common to split a data set into a training, testing, and validation set. The ratio of their sizes is generally close to 50:25:25 or 60:20:20 respectively for training, testing, and validation sets, depending on the amount of available data. The validation set is used during the training process to keep track of how well a given model learns over time as model parameters are adjusted. If validation error begins increasing while the training error is still decreasing, the model is likely beginning to overfit and training should be stopped. After training is stopped, the test set is used to measure the final performance results.

المنارة للاستشارات
www.manaraa.com

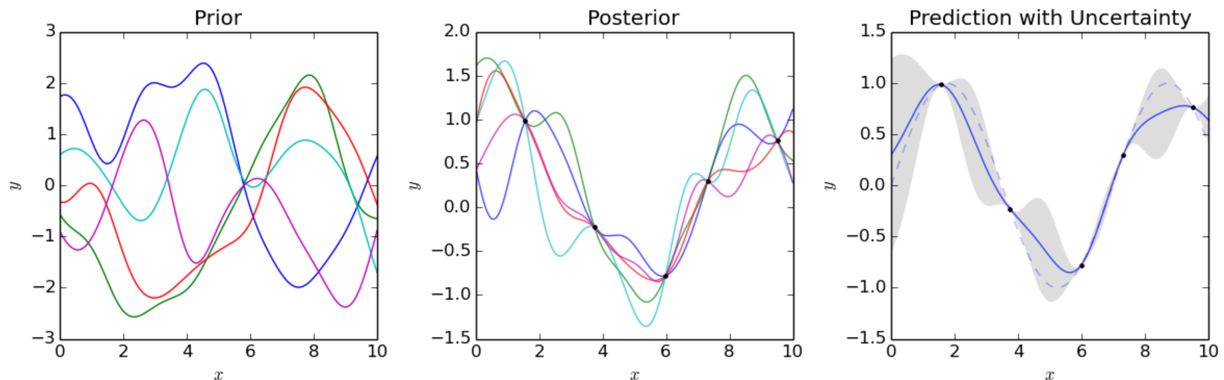

Figure 1.7 The left plot shows functions for a GPR model drawn from a prior distribution determined by the initial training data. The middle plot shows functions drawn from the posterior distribution, and the plot on the right shows mean model predictions and one standard deviation above and below for x from 0 to 10 [28].

This type of validation is a type of holdout validation since instances are removed from the initial training set to create validation and test sets.

Although this is standard for large data sets, especially when using NNs, it can be problematic for limited data sets. In limited data sets, there may only be a few instances that contain necessary information about the relationship between a specific input feature and model outputs. If these instances are randomly selected to be in the validation or testing set, it is unlikely the trained model will learn the effects of that feature. Hence, the testing error may vary greatly depending on the random subsets selected for training, validation, and testing. Ideally, all three sets will be statistically similar.

Another validation technique is k-fold cross validation. The training data are split into k equally sized subsets and a model is trained on $k-1$ subsets while the remaining subset is used as a test set. This process is repeated k times, each time using a different subset for testing. Error measures can be averaged over the k results to provide a single error metric. When k is equal to the number of training instances N, this process is called leave-one-out cross validation since only one instance is left out of the training set and used for testing each time. Leave-one-out cross validation often takes more time to calculate because the model must be trained N separate times. It can be better suited to

المنارة للاستشارات
www.manaraa.com

limited data sets because the majority of model behavior is preserved each time a separate instance is left out. However, this is not true when predictions are made on data that is statistically different to the training data. In that case, validation methods, including leave-one-out cross validation, will give optimistic error measures. There are other validation methods that split the training data in different ways, but none are well suited to validating the performance of machine learning models with limited data, especially when predictions are made on data that is statistically different to the training data set.

The Curse of Dimensionality

Although it is reasonable to believe that machine learning models would perform best when given as much information as possible, this is often not the case. When a large number of features are used, training points become sparse in feature space. Hence, a large number of features requires the availability of a large number of training instances. This is called the curse of dimensionality. Feature reduction is often performed to remove features that provide minimal information to the model and to avoid the curse of dimensionality. Further information on feature, or dimensionality, reduction is given in Sec. 2.4.

1.2.3 Uncertainty Quantification

Uncertainty quantification has existed as long as probability and statistics, and is the science of identifying, quantifying, and reducing uncertainty when predicting quantities of interest [29]. There are two main types of uncertainty, aleatoric and epistemic uncertainty [29]. Aleatoric uncertainty, or statistical uncertainty, is inherent to a problem, and hence cannot be reduced and is generally represented in terms of probabilities [29]. Epistemic, or systematic, uncertainty originates from an incomplete knowledge or missing physics in a model [29]. Six sources of uncertainty in computer models have been identified by Kennedy et. al: parameter

المنارة للاستشارات
www.manaraa.com

uncertainty, model inadequacy, residual variability, parametric variability, observation error, and code uncertainty [30].

Parameter uncertainty refers to uncertainty in parameter values that a user gives as inputs to a model [30]. In machine learning, this may be the number of hidden layers or learning rate of a NN.

Model inadequacy, or structural uncertainty, originates from uncertainty in the form of the model due to limited knowledge of the true underlying mechanisms that generate the data [30]. This form of uncertainty is also common in machine learning because each machine learning model class has a different structure. Often, the "best" models for a given problem are identified by which models produce the lowest errors, even though their structure may not match that of the true generating function.

Residual variability is comprised of two types of uncertainty [30]. The first type of uncertainty comes from the fact that the process being modeled may be inherently stochastic [30]. The second type of residual variability occurs when a model lacks the detail necessary to differentiate between two different processes [30]. Both of these sources of uncertainty can appear in machine learning when one is either trying to make predictions that are stochastic, or when more features are needed to distinguish between different model behaviors. If we want to model some variable Y with observations $y_1, y_2, ..., y_N$ and corresponding model predictions $\hat{y}_1, \hat{y}_2, ..., \hat{y}_N$, the residual variance is given by,

$$\frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2.$$

Parametric variability originates from uncertainty of inputs to the model [30]. For example, if I wanted to make a prediction from an instance that was missing some feature data, I would have to impute values for the missing fields based on feature distributions. Although no values were imputed in this thesis, this can still be a cause of uncertainty in machine learning.

Observation error, or experimental uncertainty, originates from variability of empirical measurements [30]. There will always be some amount of noise in a data set since empirical

المنارة للاستشارات
www.manaraa.com

measurements cannot be completely precise. As long as enough training data are provided with minimal observation error, the signal to noise ratio will be high enough that the general behavior of the model may still be learned.

The last type of uncertainty identified by Kennedy et al. [30], code uncertainty or interpolation uncertainty, originates from the inability to test all input configurations. Hence, it is possible to miss important model behavior because all parameter combinations were not tested. Many machine learning models can take days or weeks or longer to run if a data set is significantly large and complex. So, it would be impossible to test all parameter combinations. When possible, time should be taken to test a range of parameter combinations and identify those that are appropriate for a given data set.

1.3 Previous Research

1.3.1 Literature Review

Machine learning methods are well suited to model complex behavior when the underlying physics principles are either unknown or too complex to be modeled. In the case of predicting ambient sound levels, many physics-based principles are unknown. For example, no standard acoustic model of rivers or water sources exists currently. Mennitt et al. [31–33] used machine learning to create models designed to take geospatial features as inputs and predict various acoustic metrics. In addition, linear and nonlinear land-use regression models have been used to map urban environmental noise in northwest China [34].

Land-use models are most commonly used to model air pollution and health effects in urban areas [35]. They typically use independent variables, such as road type, traffic levels, elevation, and land cover, to create a multivariate regression model capable of predicting pollutant levels in other locations [35]. Xie et al. used acoustic data from 101 sites to train a nonlinear and linear

المنارة للاستشارات
www.manaraa.com

regression function, and used acoustic data from an additional 101 sites for validation [34]. All monitoring sites were within Dalian Municipality, Liaoning Province, China, and 36 geospatial variables were used in the training process [34]. Model predictions were mapped for the region of Dalian Municipality [34].

Mennitt, Sherill, and Fristrup used random forest models to predict ambient sound levels in 2014 [32]. A data set consisting of acoustic measurements from 190 training sites from 41 different national parks within the contiguous United States (CONUS) was used. Roughly 50 geospatial features were used as inputs to the model, and models were trained to predict seasonal one-third octave and n-percent exceeded levels. Predictions for smaller time scales, such as hourly or daytime metrics, were not explored for this data set. All seasonal measurements were utilized in the training process by using an input variable to specify the season corresponding to each instance. Leave-one-out cross validation was performed to estimate model error by leaving out all seasonal data for a given site before measuring the error for a given season at that site. The leave-one-out cross validation root-mean-square error (RMSE) and median absolute deviation (MAD) for the seasonal A-weighted L_{50} were 4.8 and 2.8 dBA respectively. The discrepancy between these two error metrics was attributed to outliers in the training data set. It was found that single-output models performed better than multiple-output models designed to predict multiple metrics simultaneously. Hence, all models used to create maps of a given acoustic metric were trained for that single metric.

Following the above study, Mennitt and Fristrup used random forests again to predict acoustic levels using a larger database in 2016 [33]. In contrast to their previous data set, acoustic measurements from 492 unique sites were used. Of those sites, 333 were located in quiet uninhabited areas within national parks and 159 sites came from urban areas. Additionally, 115 geospatial variables were used as model inputs, which can be found in Table 1 of their 2016 paper. To limit the scope of research, the daytime A-weighted L_{50} was the only predicted acoustic metric.

المنارة للاستشارات
www.manaraa.com

As in their previous report, all measured seasonal daytime A-weighted L_{50} were included in the full model. Leave-one-out cross validation was performed in the same manner as before to yield a RMSE and MAD of 4.5 dBA and 2.29 dBA respectively. Again, the discrepancy between these error metrics was attributed to outliers in the training data set.

1.4 Objectives

Mennitt et al. successfully created random forest models to make acoustic predictions [31–33]. However, there are many other types of machine learning models that may match or surpass the performance of random forests. As mentioned previously, there is no standard error metric for machine learning with limited data sets. In particular, leave-one-out cross validation may not be the most appropriate error metric for this data set due to the statistical differences between the training data and novel data on which predictions are performed. Leave-one-out cross validation can aid in identifying instances in the training data set that the model struggles to predict due to overfitting, lack of necessary distinguishing features (residual variability), or observational error (experimental uncertainty). However, it is often difficult to connect leave-one-out errors to a specific sources of error. Even if leave-one-out cross validation is used to identify areas a model struggles to make predictions, it does not yield any insight into the uncertainty of model predictions on statistically novel data.

Motivated by these limitations, this thesis extends previous results in several ways. Given acoustic and geospatial data for a wide range of locations, I generated a model using machine learning to predict ambient sound levels in environments with complex natural, biological, and anthropogenic sources. Additionally, I identified areas where model uncertainty is high. I used an ensemble of machine learning models to quantify the structural uncertainty and estimate model performance on instances that are statistically different from data represented in the training set. I

المنارة للاستشارات
www.manaraa.com

also used the ensemble of models to make predictions of acoustic metrics that vary in frequency, time, and space. Results were generated using a computational pipeline, which preprocesses and loads data, trains the ensemble model, and makes predictions for various acoustic metrics. To visualize results, maps of ensemble model predictions and uncertainties for regions in CONUS were produced.

للاستشارات المنارة
www.manaraa.com

Chapter 2

Methodology

2.1 Data Sets

Acoustic and geospatial data was obtained and organized by Blue Ridge Research and Consulting (BRRC) for sites in CONUS. Acoustic data sources include the NPS [33], BRRC, the Environmental Protection Agency (EPA) [36], and a trusted third-party consulting firm. BRRC checked the quality of all measurements and removed any that were found to be unsatisfactory due to excess wind noise, instrumentation error, etc. BRRC then consolidated this into a database that contains acoustic data for 607 training sites. The acoustic data contain several statistical noise levels for each season, such as the L_{50} and L_{90}, and the equivalent level L_{eq}. The database also contains these statistical noise levels in three specific frequency bands (12.5-125 Hz, 160-1,250 Hz, 1,600-12,500 Hz) and one-third octave bands. For each season, acoustic metrics are provided for hourly, daytime, and nighttime temporal domains. Data are not generally available at all sites for all seasons and metrics. However, most sites have summer acoustic metrics, so I focused my research on summer data. Summertime L_{50} measurements are available for 502 of the 607 training sites.

المنارة للاستشارات
www.manaraa.com

This data set is different from those that Mennitt et al. used [32,33]. It does not include acoustic measurements from the 145 airport noise monitoring system locations [37]. However, our data set contains an additional 84 training sites from the NPS, 54 from BRRC, 100 from the EPA, and 22 from a trusted third-party consulting firm. Of all 431 training sites from the NPS, only 326 contain summer acoustic metrics. A summary of summertime acoustic metrics is provided in Appendix B.

In addition to acoustic data, 117 geospatial feature measurements were available for almost the entire CONUS region from a NPS inventory. These include features such as the distance to the nearest road and airport, the distance to the nearest body of water, and the average temperature. A table of all geospatial features is provided in Appendix A. Figure 2.1 and Figure 2.2 show maps of two geospatial inputs: mean upward radiance at night (270 m resolution) and proportion of forest landcover (200 m resolution). These are just two of the 117 geospatial features that serve as inputs to the machine learning models.

The geospatial database is very similar to that used by Mennitt and Fristrup in their 2016 paper [33]. Because they included all seasonal predictions in the training process, they had input variables to distinguish between different seasons. I limited my research to summertime acoustic metrics and did not utilize other seasonal acoustic data, so it was not necessary for me to include a feature to specify the season. Mennitt and Fristrup also included the following features: day length, road density, proportion of snow landcover, annual mean wind speed at 50 m above ground, and a categorical topographic position index. I anticipate that most of these features are unlikely to significantly affect any machine learning model. For example, snow landcover is minimal within CONUS during the summer, and it is also unlikely that training points exist in areas where there is snow landcover. Hence, the feature would provide little, if any, new information during the training process.

المنارة للاستشارات
www.manaraa.com

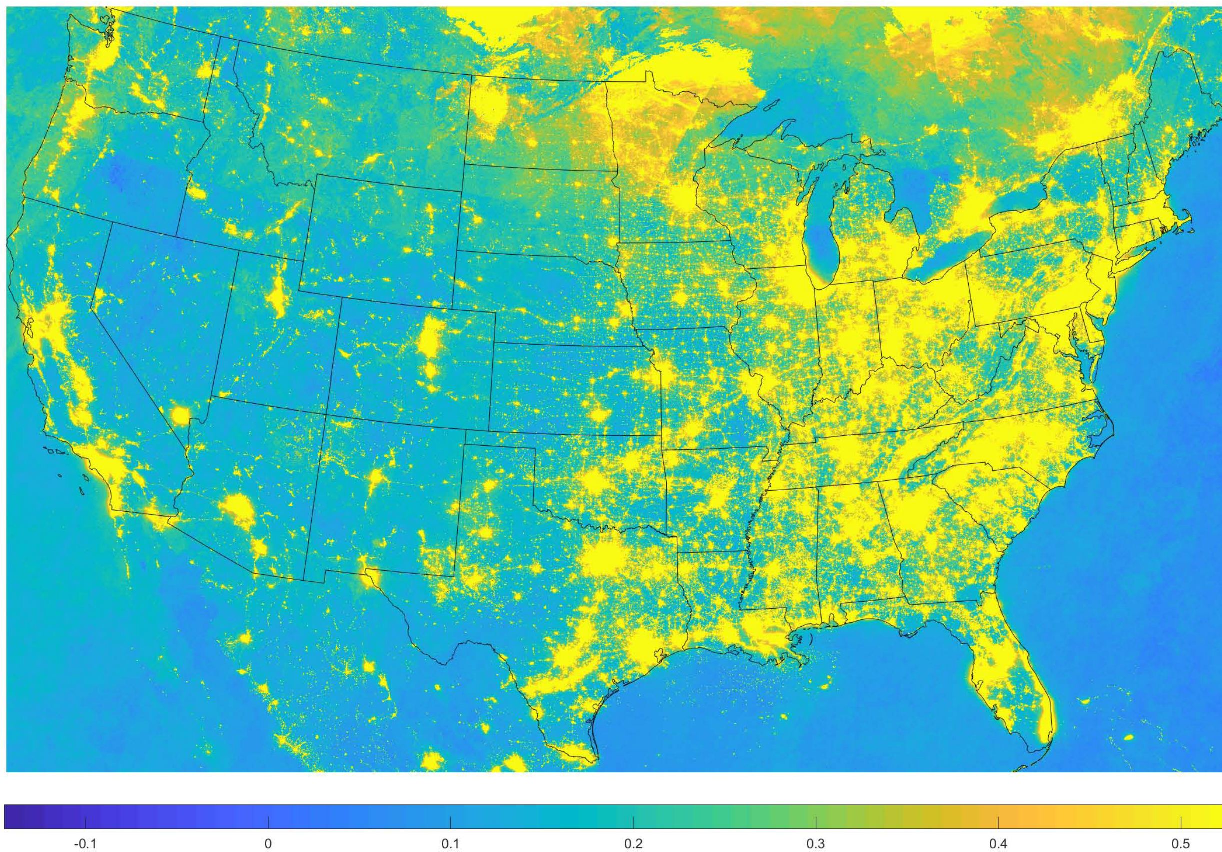

Figure 2.1 Mean upward radiance at night with a 270 m resolution. This is one of the geospatial inputs used in machine learning. Units are nW/cm^2/sr.

للاستشارات المنارة
www.manaraa.com

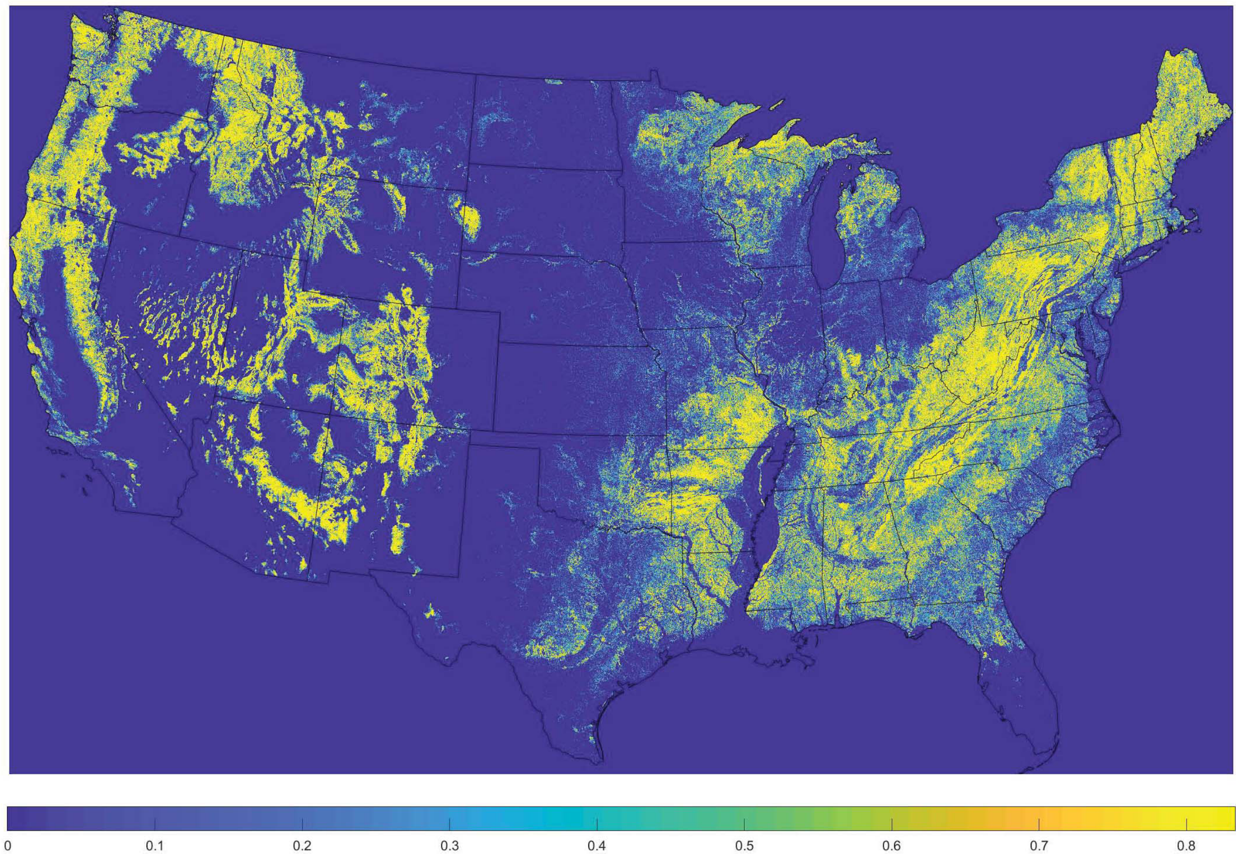

Figure 2.2 Proportion of forest landcover with a 200 m resolution. This is one of the geospatial inputs used in machine learning.

المنارة للاستشارات
www.manaraa.com

2.2 Computational Pipeline

A computational pipeline was created to increase efficiency and organization. Code for the pipeline, which was implemented in Python, is included in Appendix C. Figure 2.3 shows a flowchart of the pipeline. First, the training data for all acoustic metrics and geospatial features is loaded. The user must specify which acoustic metrics they would like to make predictions for. Additionally, the user must specify a scaler, or transform, to use on the acoustic and geospatial data, such as the identity scaler or standard scaler, which normalizes all features to have zero mean and a standard deviation of 1. The user may also choose a specific set of features with which to train and make predictions if they do not want to use the complete geospatial data set. After the data has been preprocessed to meet the specifications of the user, all members of the ensemble model are trained. Then, predictions are made and saved for all members of the ensemble. If any model runs into undefined geospatial inputs while making predictions, it will leave the prediction undefined.

The process of making model predictions has been parallelized to speed up run time. However, the ensemble training and predicting process for a single acoustic metric and all of CONUS still takes roughly five and a half hours using 50 cores and approximately 250 GB RAM.

In addition to the acoustic metric(s), scaling method, and geospatial features, the user may also specify the number of cores to be used when making predictions, the specific models to be trained, and the regions of CONUS (northwest, northeast, southwest, and southeast) over which to make predictions. The pipeline is also capable of conducting various feature importance and validation measures. This is not meant to be an exhaustive description of all pipeline functionalities. For more detail, refer to Appendix C.

المنارة للاستشارات
www.manaraa.com

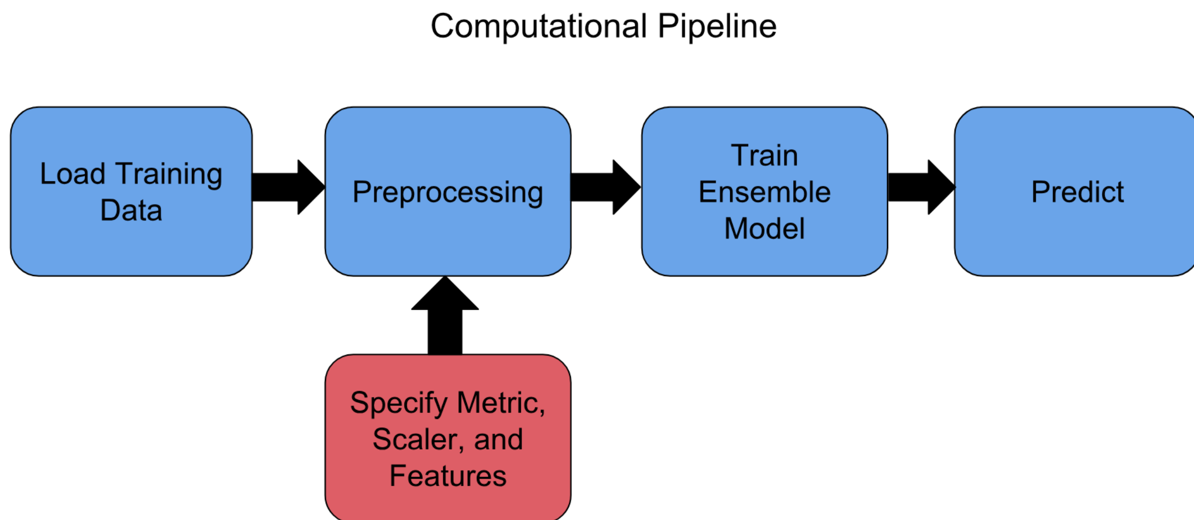

Figure 2.3 Flowchart showing the general process used by the computational pipeline to make predictions.

2.3 Model Selection and Parameter Tuning

The computational pipeline made it easy to measure various validation metrics for a wide variety of machine learning models. To select models for the ensemble and also identify appropriate parameter values for these models, I explored six machine learning algorithms, GBR, NNs, KNN, SVMs, KR, and GPR (see Sec. 1.2.2). I used the scikit-learn library in Python to implement these algorithms [38]. A variety of parameter combinations were tested for each model. To compare initial model performance, the leave-one-out cross validation was used to calculate the RMSE and MAD for each model. Parameters were tuned for each model to minimize the leave-one-out MAD.

2.4 Feature Reduction

I made a first attempt at feature reduction to combat the curse of dimensionality when using a limited training data set. I used four different metrics of feature importance to systematically

المنارة للاستشارات
www.manaraa.com

reduce the number of dimensions of the data set.

The first metric used is the Gini importance or mean decrease impurity [27]. This is a common way to measure feature importance in a random forest or GBR. The basics of the calculation of the Gini importance metric for classification are given here, but the reader is encouraged to reference Breiman's article [27] or Marsland's text [24] for further information. If we let N_i be the fraction of data points belonging to class i, then a pure leaf or node of class j should have $N_j = 1$ and $N_{i,i \neq j} = 0$. The Gini impurity for some feature k and c classes is defined by,

$$G_k = \sum_{i=1}^{c} \sum_{j \neq i} N(i)N(j).$$

Since $\sum_{j \neq i} N(j) = 1 - N(i)$,

$$G_k = 1 - \sum_{i=1}^{c} N(i)^2.$$

In other words, the Gini impurity is the expected error rate if classification was selected according to the distribution of classes. Hence, features with low Gini impurity have the highest Gini importance.

The second metric used for feature importance used the Gini importance metric with a correlation penalty. Note that if two features are identical, they will have the same Gini importance measures, but the model does not need both features. A correlation penalty was applied to the Gini importance to help avoid high levels of correlation among the top ranked features. The Gini importance was first calculated for all features. Then, for any given feature, I found the feature from the remainder of the data that was most highly correlated with that feature. From those two features, I selected the one with the lowest Gini importance and subtracted the correlation times its Gini importance. If two features were perfectly correlated, one would have an importance of 0 and the other would have the same importance given by the original Gini metric.

The third metric used for feature importance is only applicable to NNs. There is no standard way to measure feature importance in a NN, but many methods have been suggested [39]. I chose

المنارة للاستشارات
www.manaraa.com

to use the weights to measure feature importance. First, I identified all paths from an input feature to the output, and calculated the product of all weights along each path. Then, for each feature, I summed the absolute value of all paths originating at that feature. Finally, these sums were normalized and the results were used as a feature importance measure. For the case of zero hidden layers, the feature importance was determined by the magnitude of the weights from the input features to the output.

My last metric required someone to look at each geospatial input feature and remove them one at a time, based on their human intuition for what information the model would require. Features that were removed early generally had lower spatial resolutions or contained minimal information. Additionally, features that were highly correlated with other features were removed early because they provided no new information.

All feature importance lists presented in the results section were created by iteratively measuring the feature importance values and removing the feature with the lowest importance. Features were ranked according to the order in which they were removed. Note that these results are not identical to measuring the feature importance values once with all features as model inputs. The act of removing features will often shift the order of importance of other features.

المنارة للاستشارات
www.manaraa.com

Chapter 3

Results

3.1 Ensemble of Models

3.1.1 Model Selection

After tuning all hyperparameters and parameters for a variety of machine learning models to minimize the leave-one-out MAD, the leave-one-out MAD and RMSE were examined. Figure 3.1 shows a histogram of the residuals created from leave-one-out cross validation for the KNN model. Looking at these residuals (of the measured value and leave-one-out predicted value), we see that the distribution is generally not Gaussian. The outliers explain why the MAD is normally half the value of the RMSE (see Table 3.1). Similar behavior is seen in the literature by Mennitt et. al [32, 33].

As seen in Table 3.1, all leave-one-out MAD and RMSE values are within 1 dBA of each other and 1 dBA is typically the smallest change in SPL that is perceptible to a human. So, it is reasonable to conclude that all six models performed similarly, and should all be included in the ensemble model. To reduce sensitivity to any one model predicting extremely small or large results, the median value of the ensemble was used rather than the mean. Using the median,

المنارة للاستشارات
www.manaraa.com

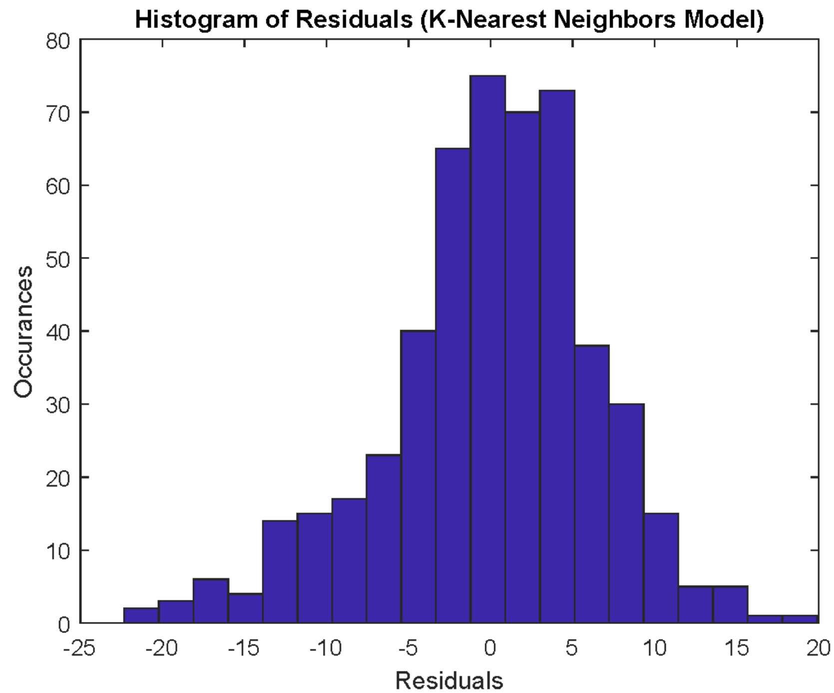

Figure 3.1 Histogram of the residuals from performing leave-one-out cross validation using a KNN model.

rather than the mean, helps stabilize ensemble predictions when predictions are made for novel and unique instances. Note that it would not have been unreasonable to use the mean ensemble prediction rather than the median. When the standard deviation of ensemble model predictions is low, I expect the median and mean to have similar values.

Model Class	Leave-One-Out MAD (dBA)	Leave-One-Out RMSE (dBA)	Fit MAD (dBA)	Fit RMSE (dBA)
GBR	3.5	6.0	0.08	1.2
NN	3.7	6.3	3.4	5.7
KR	3.6	6.3	0.3	1.4
KNN	3.7	6.6	0.0	1.2
GPR	3.6	6.2	2.1	4.0
SVM	3.4	6.2	1.3	4.2

Table 3.1 Fit and leave-one-out cross validation errors for six different machine learning models.

Table 3.1 shows that all models have similar leave-one-out cross validation errors, even though

المنارة للاستشارات
www.manaraa.com

some of their fit errors are fairly different. Fit errors help tell us which models are overfitting, such as the GBR model, and which are not, such as the NN. A very shallow NN (with no hidden layers) was chosen to avoid overfitting, so it is not surprising that the full model performs poorly on the training data.

3.1.2 Ensemble Advantages

One advantage of using an ensemble model is that it provides some measure of uncertainty in model predictions in locations that are statistically different to the training set. Figures 3.2, 3.3, and 3.4 show maps of CONUS predictions for the summer daytime L_{50} sound levels using the GBR model, NN model, and ensemble model respectively. The small circles on the maps denote the locations of all training sites. Figure 3.5 shows the standard deviation of ensemble model predictions.

GBR model predictions look reasonable, but I have no method of assigning confidence intervals to model predictions. The NN model predictions look unphysical in several places, suggesting large model uncertainties in such areas. NN model predictions suggest that much of Iowa, Illinois, and the surrounding states are quite loud. Additionally, there are some circular areas, such as in eastern Montana and western Texas, that show low SPLs surrounding large SPLs. It is possible, though unlikely, that these model predictions are correct, but there is no way to know without traveling there and measuring the SPLs. GBR, NNs, and several other machine learning models do not have confidence intervals or error bars on their predictions. However, the ensemble model predictions when combined with the standard deviation of model predictions gives us a means of quantifying the uncertainty of each prediction. The confidence intervals provided by the standard deviation of ensemble predictions are similar to those generated by a GPR model, which also uses an ensemble of models, or functions.

The additional information provided by the standard deviation of ensemble models can help

المنارة للاستشارات
www.manaraa.com

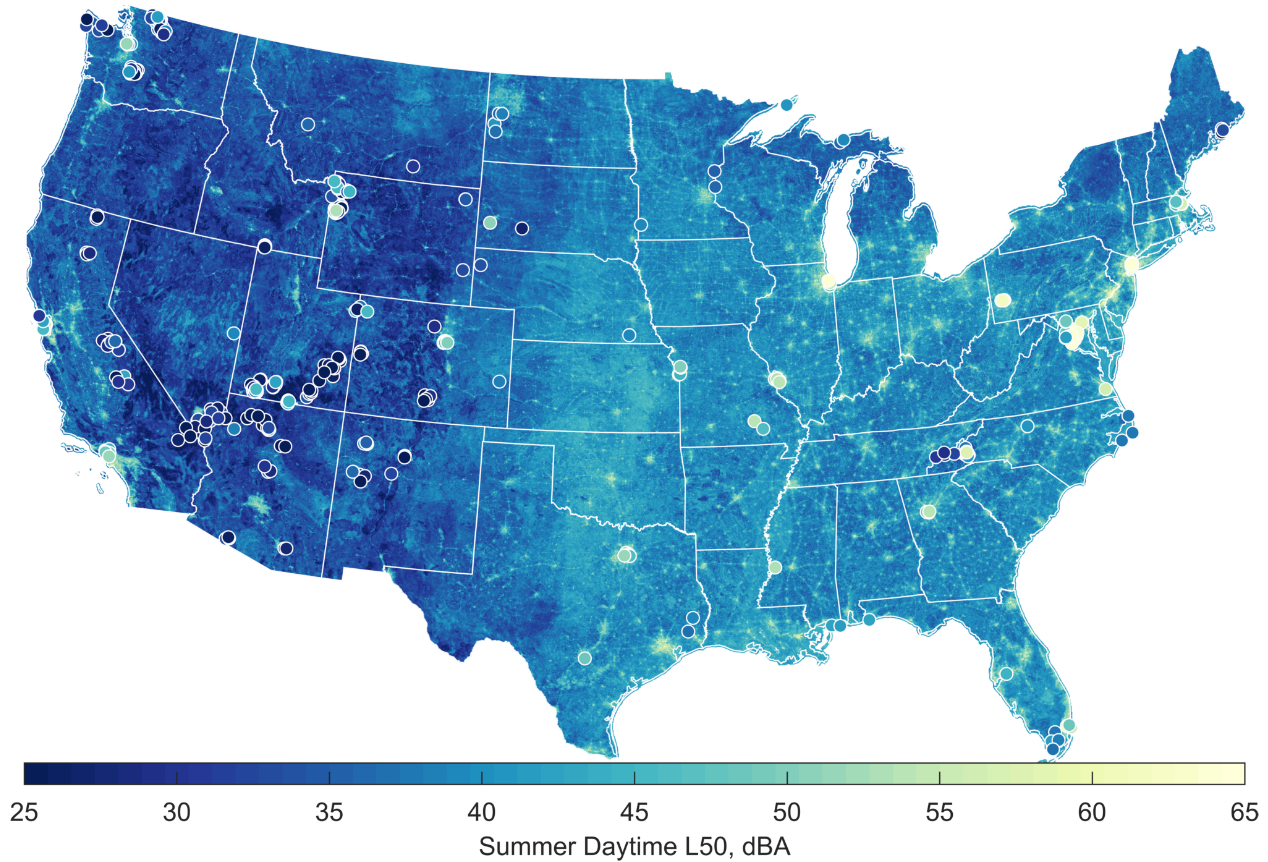

Figure 3.2 GBR model predictions for the A-weighted summer daytime L_{50} for CONUS. Training sites are marked by small circles.

identify regions of large uncertainty. Note that the ensemble standard deviation is a measure of structural uncertainty since all models that are members of the ensemble have a different inherent structure. The availability of standard deviation values can direct some efforts to improve the current training data set because I can focus data collection in areas of higher uncertainty. Areas of higher uncertainty presumably correspond to underrepresented areas of the training data set. The area around Iowa and Illinois, and the unusual circular regions where NN predictions look suspicious are all places of higher standard deviation. This information is useful for improving the current ensemble model.

المنارة للاستشارات
www.manaraa.com

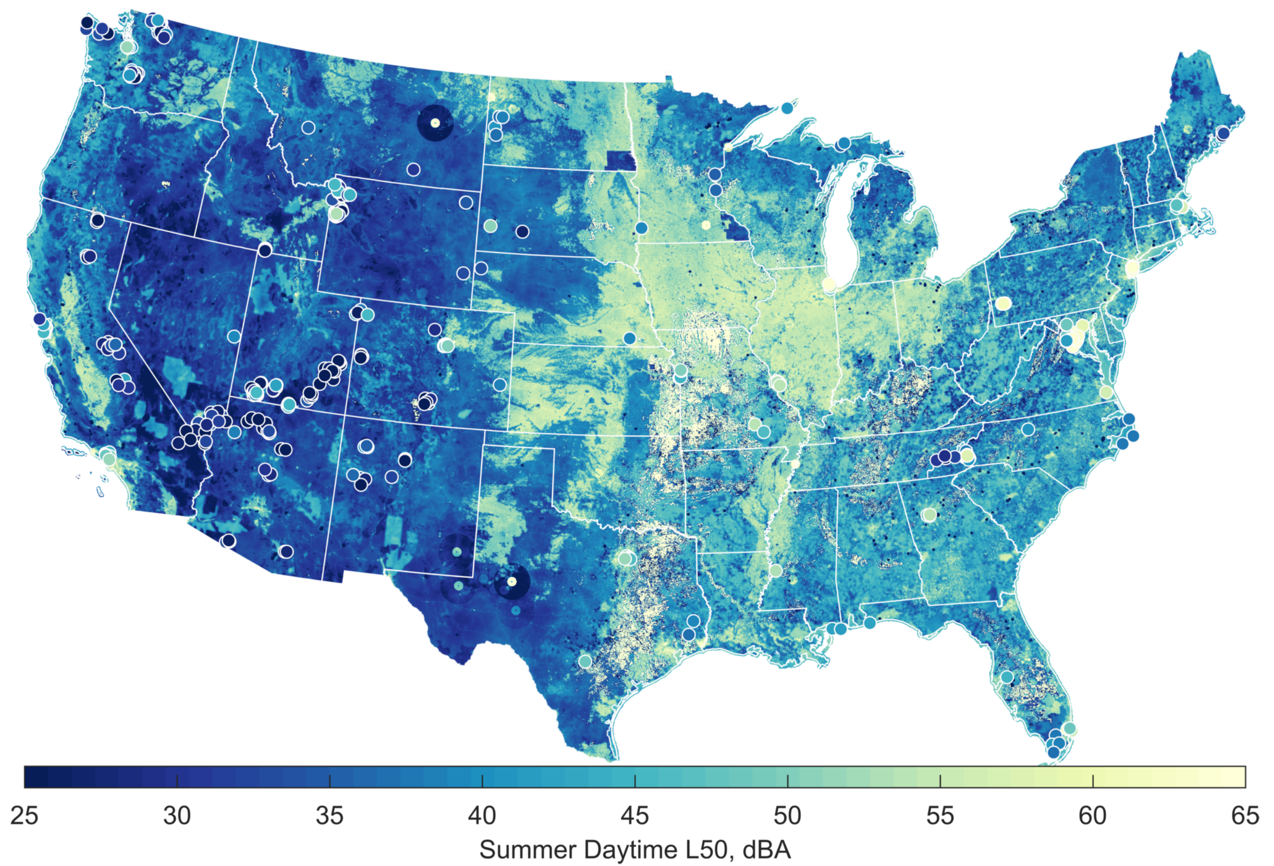

Figure 3.3 NN model predictions for the A-weighted summer daytime L_{50} for CONUS. Training sites are marked by small circles.

المنارة للاستشارات
www.manaraa.com

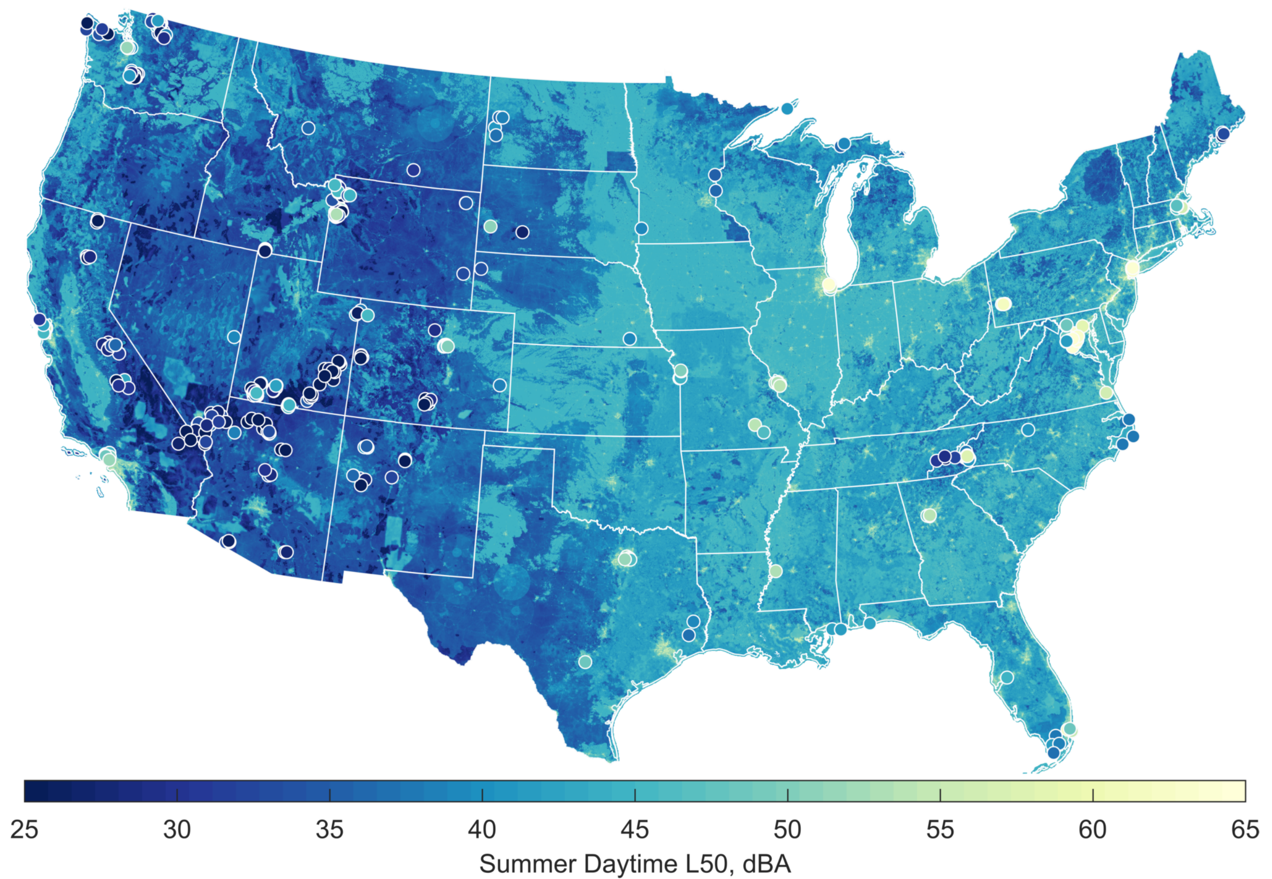

Figure 3.4 Ensemble model predictions for the A-weighted summer daytime L_{50} for CONUS. Training sites are marked by small circles.

المنارة للاستشارات
www.manaraa.com

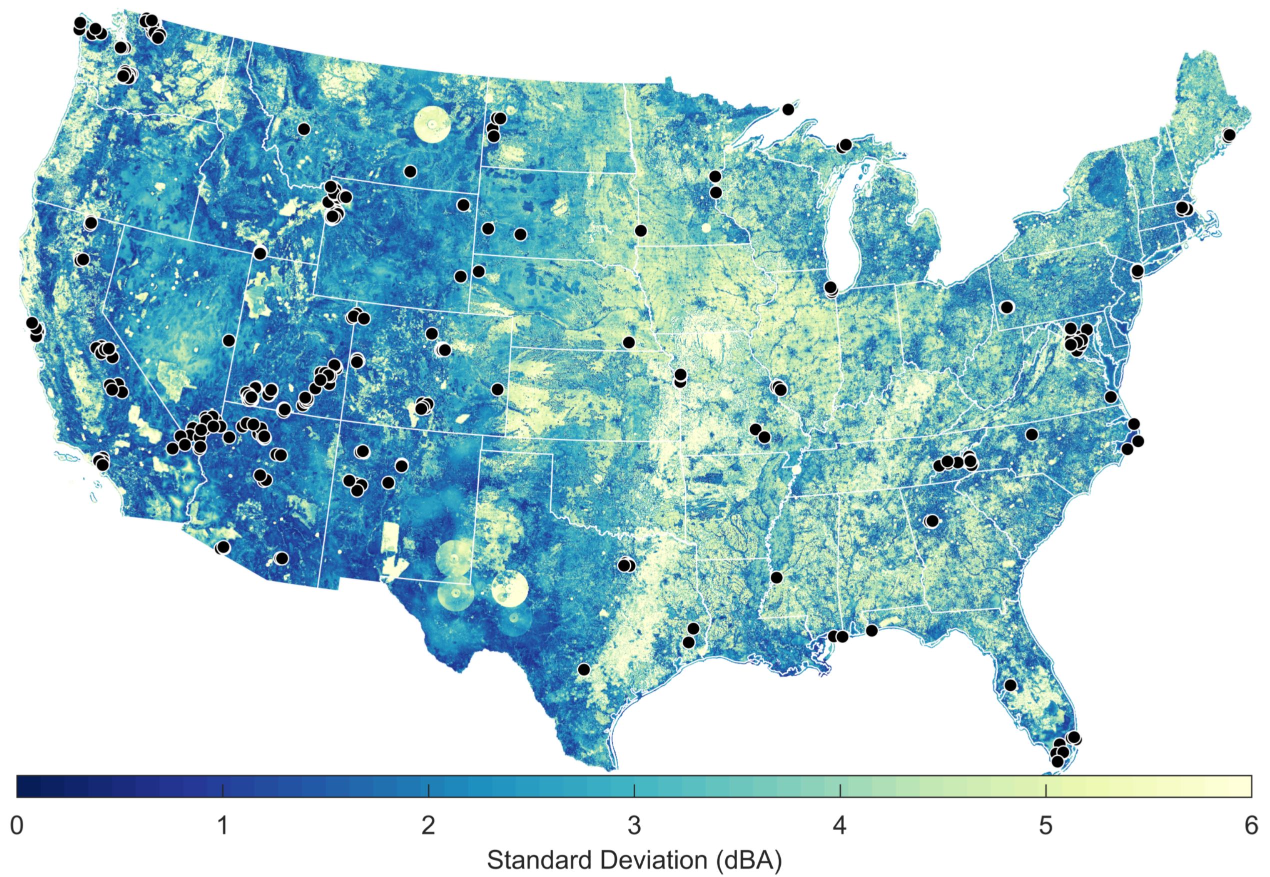

Figure 3.5 Standard deviation of ensemble model predictions for the A-weighted summer daytime L_{50} for CONUS. Training sites are marked by small circles.

المنارة للاستشارات
www.manaraa.com

3.2 Maps

3.2.1 Summer L_{10}, L_{50}, and L_{90} Day and Nighttime Maps

Ensemble model predictions were made for the A-weighted L_{10}, L_{50}, and L_{90} metrics for both day and nighttime. The results are shown in Figure 3.6. Levels are on average higher during the day, as one would expect since there is a greater contribution of anthropogenic noise in the daytime. Additionally, the L_{10} is typically higher than the L_{50}, which is typically higher than the L_{90}, as expected.

It is curious that the eastern half of CONUS is significantly louder than the western half on average. Although I am not ruling out the possibility that this is physical, I believe this to be an artifact of the distribution of our limited training set. Most of the urban (and hence louder) sites are in the eastern CONUS area, while most NPS (or quieter) sites are in the western CONUS area. The ensemble model has likely learned this and therefore expects all parts of the eastern CONUS area to be louder. Further investigation and data are needed to determine if the eastern CONUS area is actually louder, but the standard deviation maps do provide some information as to the uncertainty in those predictions.

In Figure 3.7, I have mapped the standard deviation of the ensemble predictions. It is clear that there is a lot more variation and disagreement among the models during the night primarily in the area from eastern Texas north to Iowa. The scale on the standard deviation maps ranges from 0 to 20 dBA, which is extremely large. These maps help identify areas of large uncertainty, which are good candidates for additional data collection.

3.2.2 Frequency Group Maps

I also predicted sound levels for three frequency groups (12.5-125 Hz, 160-1,250 Hz, and 1,600-12,500 Hz). Engine and surf noises are in the low frequency group, aviation and wind noise are

المنارة للاستشارات
www.manaraa.com

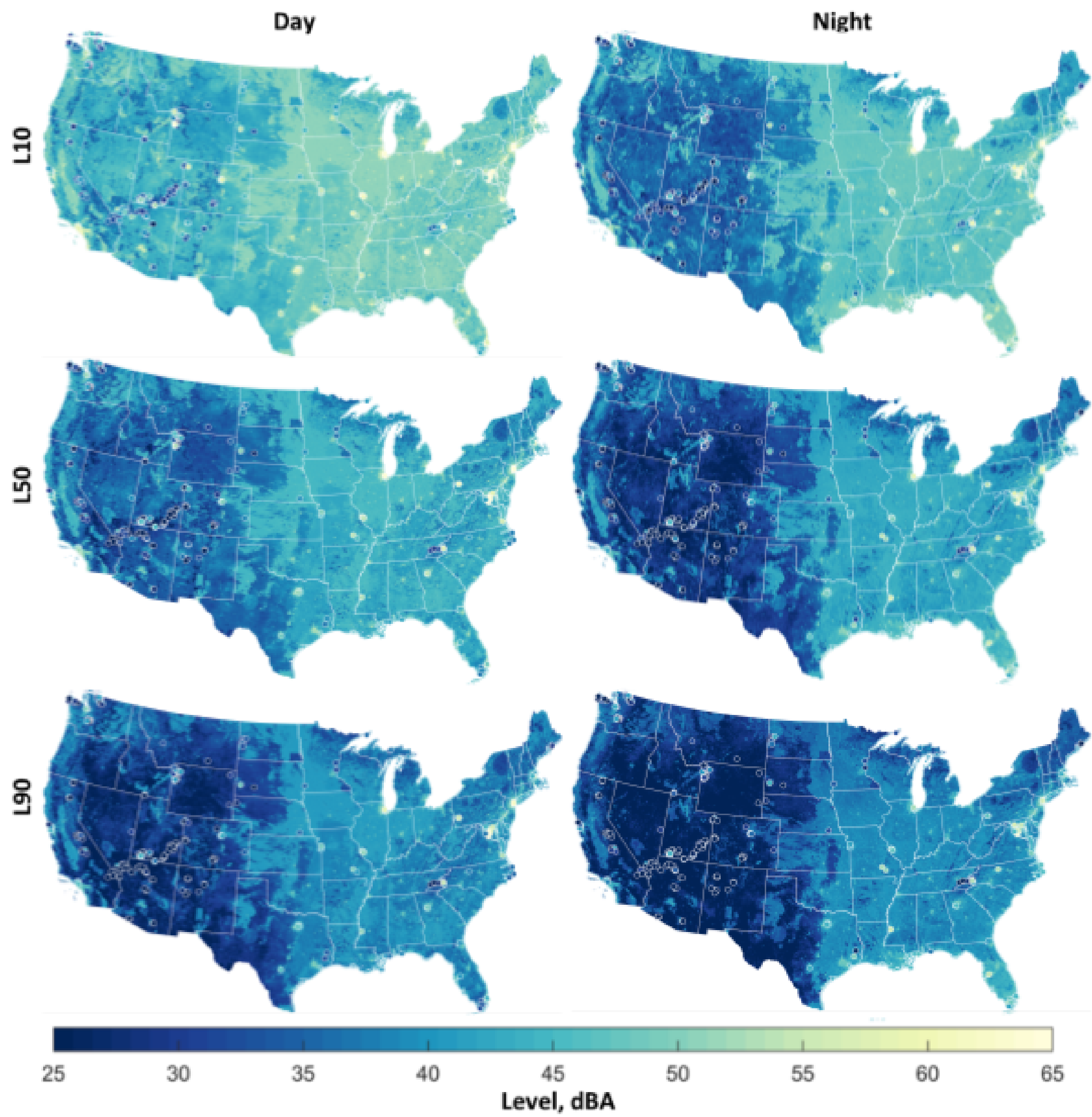

Figure 3.6 Ensemble model predictions for the A-weighted L_{10}, L_{50}, and L_{90} for day and nighttime.

المنارة للاستشارات
www.manaraa.com

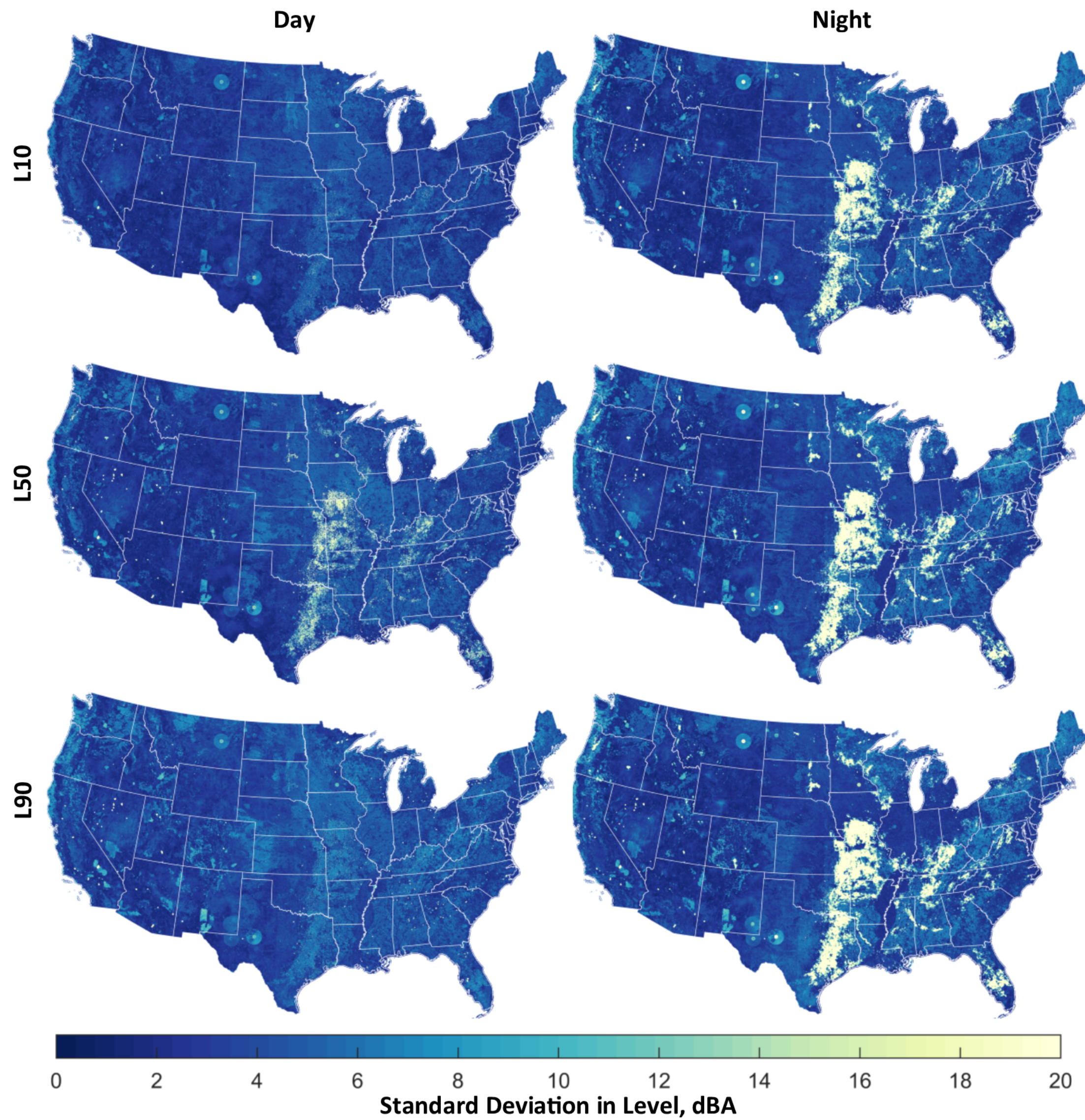

Figure 3.7 Standard deviations of the ensemble model predictions for the A-weighted L_{10}, L_{50}, and L_{90} for day and nighttime.

المنارة للاستشارات
www.manaraa.com

in the middle frequency group, and insect and bird noise are in the high frequency group. Maps of the ensemble-predicted daytime flat-weighted L_{50} sound levels for western North Carolina and Asheville are shown in Figure 3.8. The training points and their measured sound levels are shown as small circles on the map. Although not all training points are in agreement with the ensemble model predictions, most are in very close agreement.

3.2.3 Hourly Summer L_{50} Frequency Group Maps

Ensemble model predictions were made for the flat-weighted summer L_{50} for all frequency groups and hours in the Asheville, North Carolina area. Looking at all hourly frequency group ensemble predictions, I observed that daytime levels are usually higher and more variable than nighttime levels. Additionally, road noise is most prominent during the day, starting during the morning commute around 7 a.m., and persisting until roughly 6 p.m. I also found that high frequency noise tends to be localized around highways during the day, but become less restricted around 10 p.m. I suspect that this non-local increase in high frequency noise is due to insect noise.

To help illustrate some of these conclusions, figures of the Asheville area are shown for the summer L_{50} low (12.5-125 Hz) and high (1,600-12,500 Hz) frequency groups at 4 a.m., 8 a.m., and 10 p.m. (see Figures 3.9, 3.10, and 3.11). Low and high frequency group ensemble predictions are shown on the left and right respectively. The circles on the maps again correspond to training sites and their measured values. Predictions are provided at 4 a.m. as a representation of average levels throughout the night. Levels are higher during the 8 a.m. hour, particularly around roads for both the low and high frequency groups. This is likely due to rush hour traffic from the morning commute. Nighttime levels in general are less pronounced along major roads, representing a decrease in anthropogenic noise. Levels at 10 p.m. are relatively large in the high frequency group, but are more widespread than during the morning commute. This spatial dependence suggests that the increase in high frequency noise near 10 p.m. is possibly due to insect noise. Note that

المنارة للاستشارات
www.manaraa.com

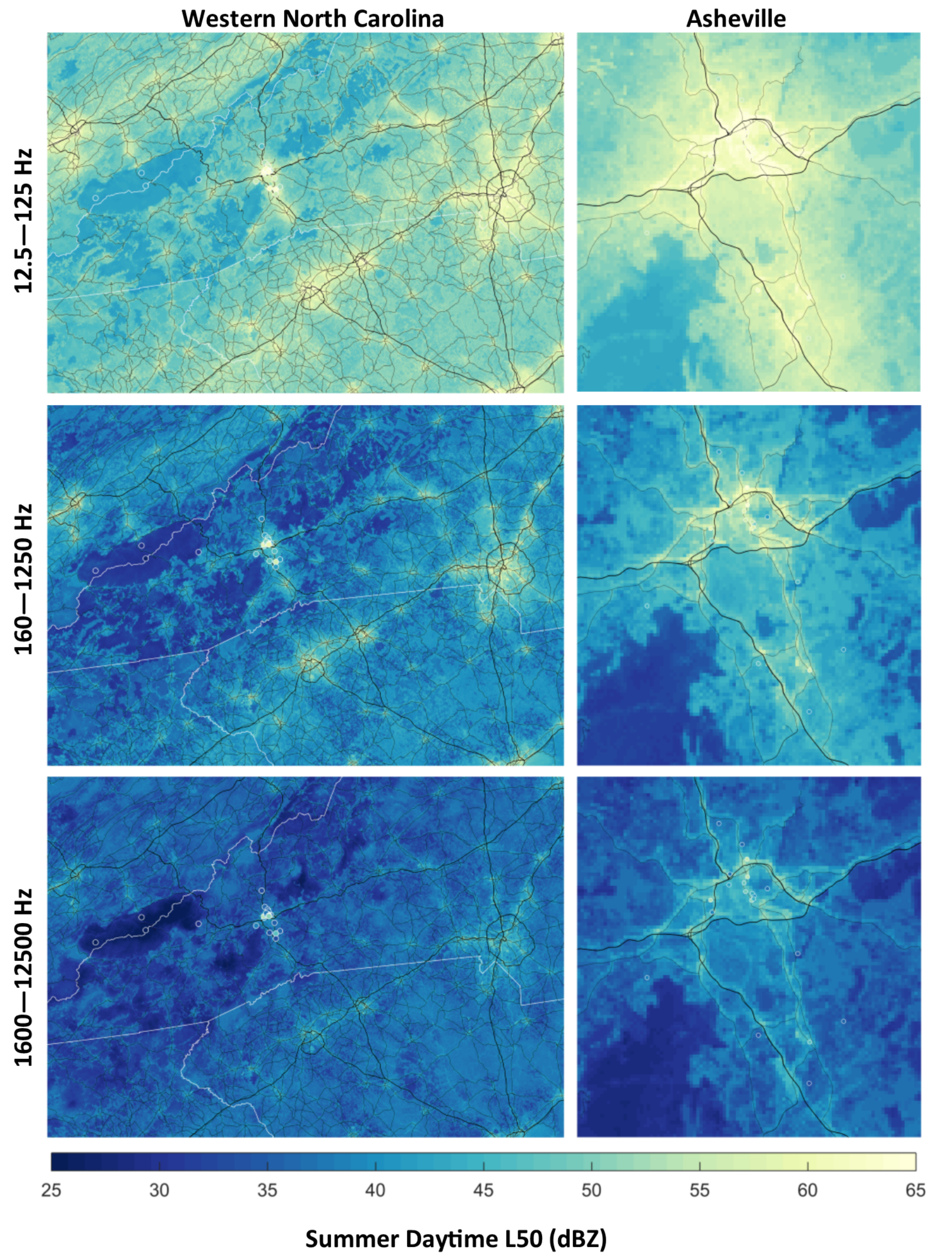

Figure 3.8 Ensemble model predictions for the flat-weighted L_{50} summer daytime frequency groups. The maps on the left show the western North Carolina area and the maps on the right are zoomed in on Asheville. The lowest frequency group maps are on the top and the highest frequency group maps are on the bottom.

المنارة للاستشارات
www.manaraa.com

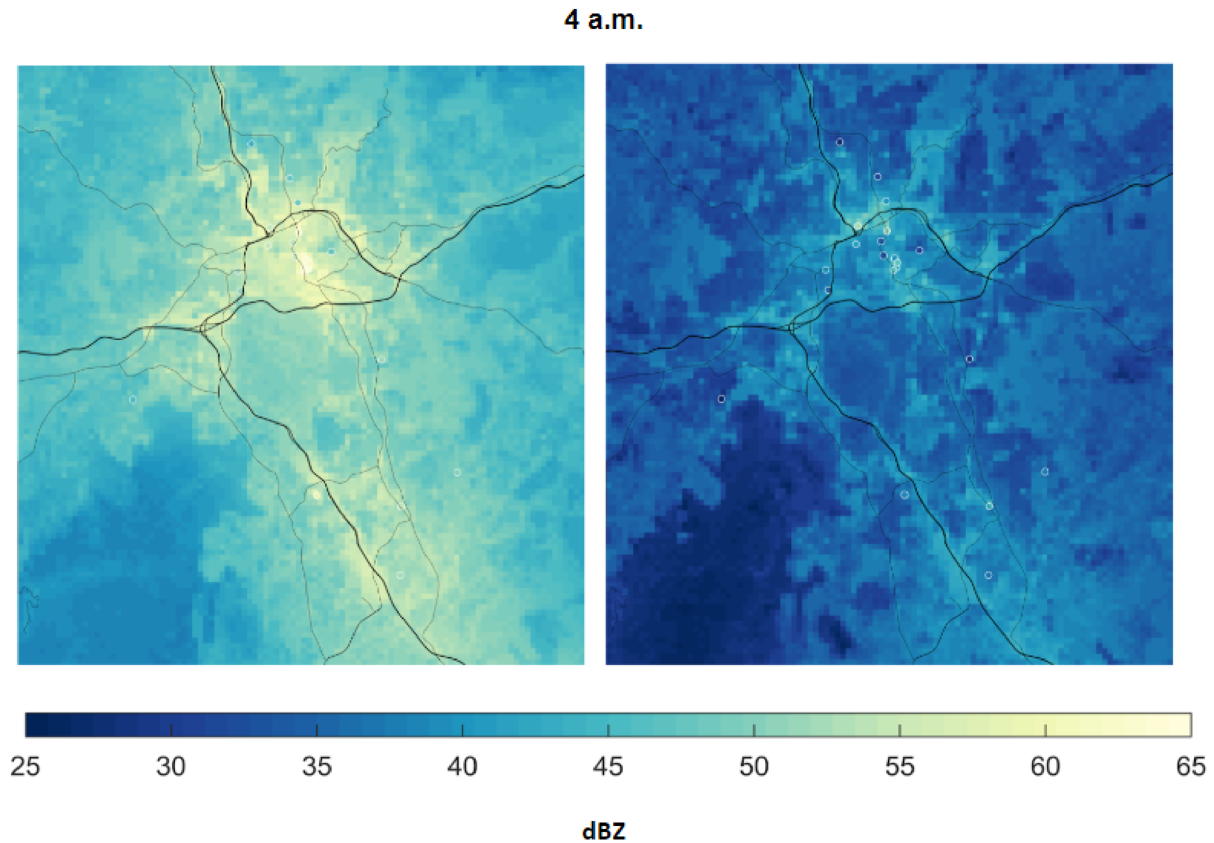

Figure 3.9 Ensemble model flat-weighted L_{50} summer predictions for the Asheville, North Carolina area at 4 a.m. for the low (12.5-125 Hz) and high (1,600-12,500 Hz) frequency groups shown on the left and right respectively.

ensemble predictions do a decent job predicting levels at the training sites in all maps. This is promising and suggests that the model will perform well if enough statistically similar training data (when compared to new input sites) are obtained.

3.3 Leave-Four-Out Validation Study

Four sites were selected to simultaneously remove from the training data set as part of a validation study. The four sites were chosen to be unique from one another and illustrate some strengths and weaknesses of the ensemble model. In no particular order, the removed sites were from (1)

المنارة للاستشارات
www.manaraa.com

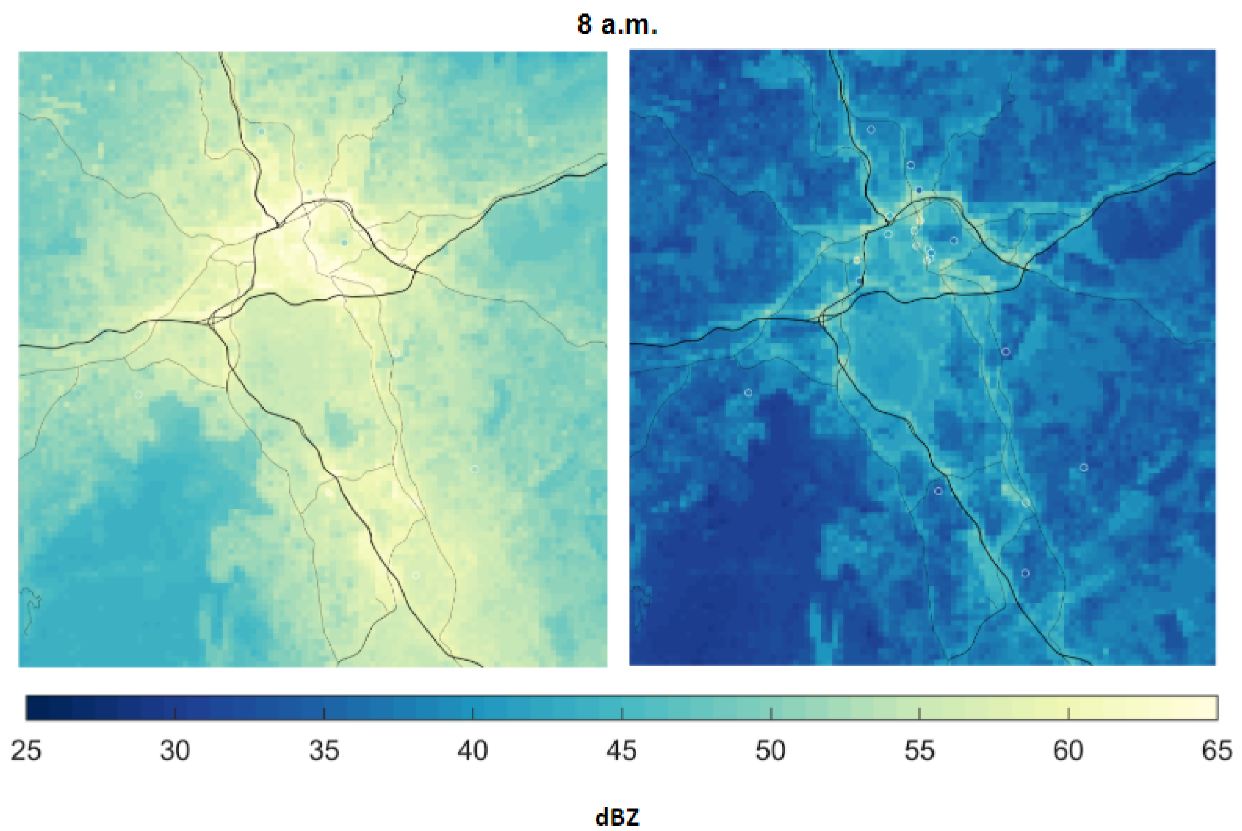

Figure 3.10 Ensemble model flat-weighted L_{50} summer predictions for the Asheville, North Carolina area at 8 a.m. for the low (12.5-125 Hz) and high (1,600-12,500 Hz) frequency groups shown on the left and right respectively.

المنارة للاستشارات
www.manaraa.com

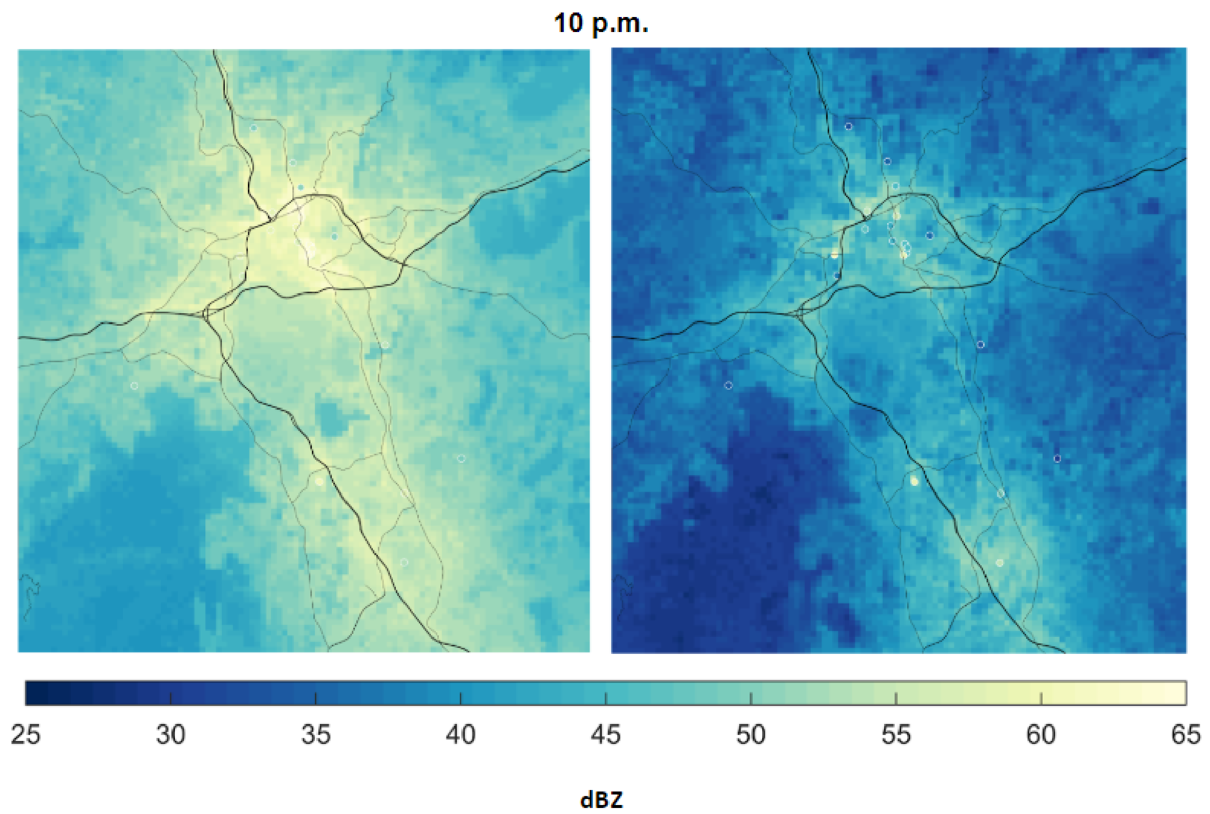

Figure 3.11 Ensemble model flat-weighted L_{50} summer predictions for the Asheville, North Carolina area at 10 p.m. for the low (12.5-125 Hz) and high (1,600-12,500 Hz) frequency groups shown on the left and right respectively.

المنارة للاستشارات
www.manaraa.com

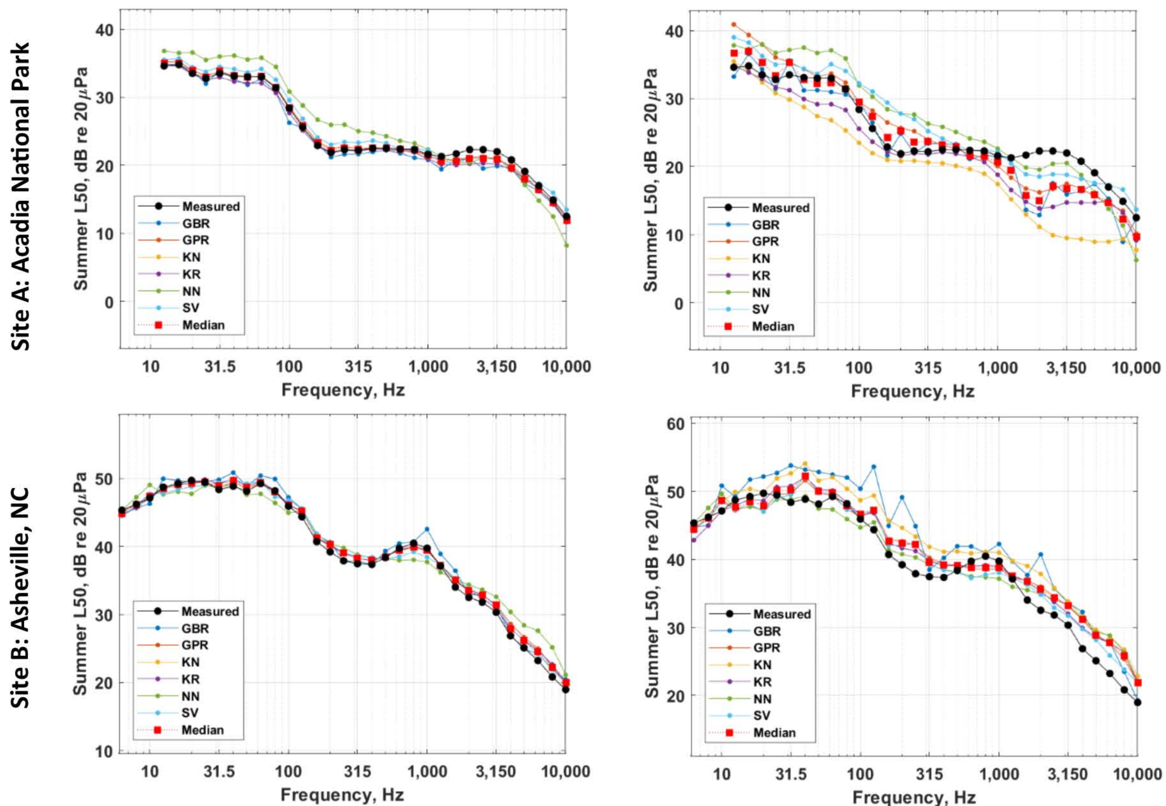

Figure 3.12 Predictions and measured levels at the first two of four chosen validation sites as a function of frequency. Plots on the left show daytime model predictions when the complete training data set is used. Plots on the right are similar, but were created from predictions using a leave-one-out training data set. In other words, all models were trained using the complete training data set, excluding the site of interest.

Gilmore Meadow in Acadia National Park, (2) a private home in a residential area in Asheville, NC, (3) the Fairfax Circle Shopping Center in Virginia, and (4) about one mile northeast of the National Mall near a railroad in Washington, D.C. A large portion of the training data set is from national parks, so the first site was chosen from national park data. Notes in the training data set indicated that the second site has significant insect noise at night. There was interest in comparing high frequency predictions and measurements there. The third site has a significant amount of road traffic noise, and the fourth site should have noise contributions due to railroad trains and nearby MetroRail trains.

Figures 3.12–3.15 show the day and nighttime model predictions for all machine learning

المنارة للاستشارات
www.manaraa.com

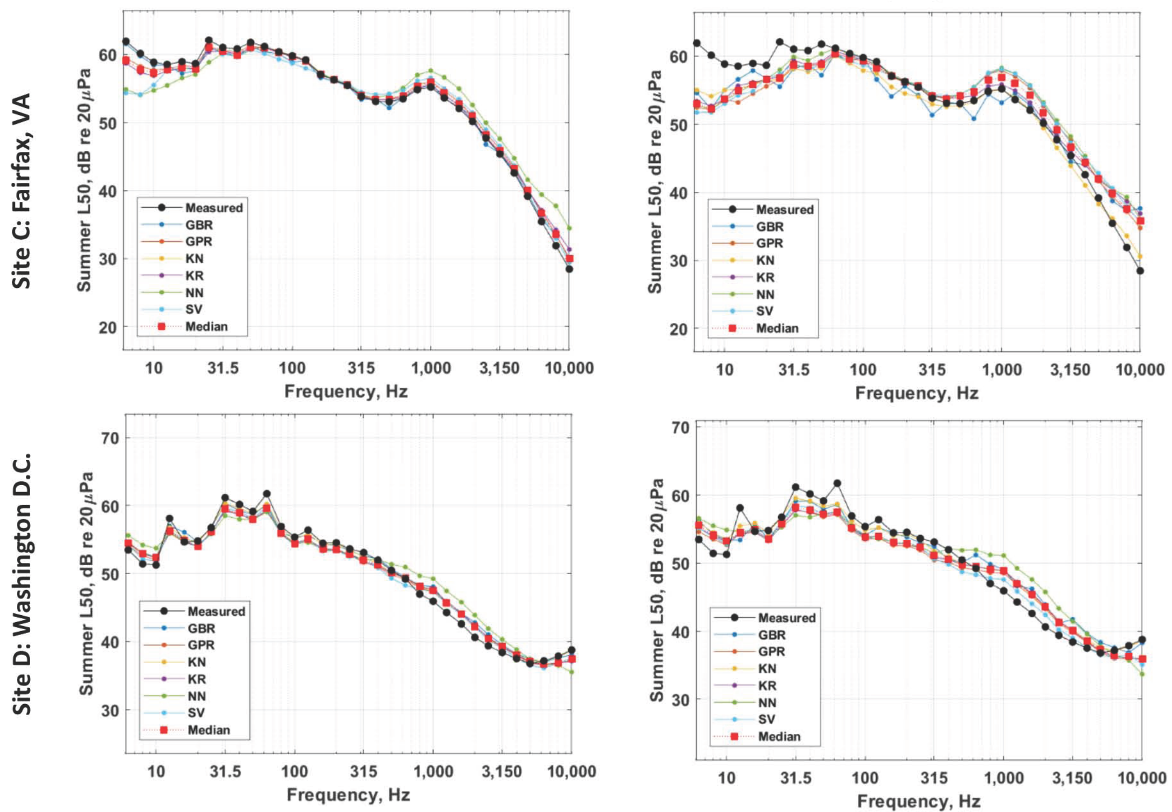

Figure 3.13 Predictions and measured levels at the last two chosen validation sites as a function of frequency. Plots on the left show daytime model predictions when the complete training data set is used. Plots on the right are similar, but were created from predictions using a leave-one-out training data set. In other words, all models were trained using the complete training data set, excluding the site of interest.

www.manaraa.com

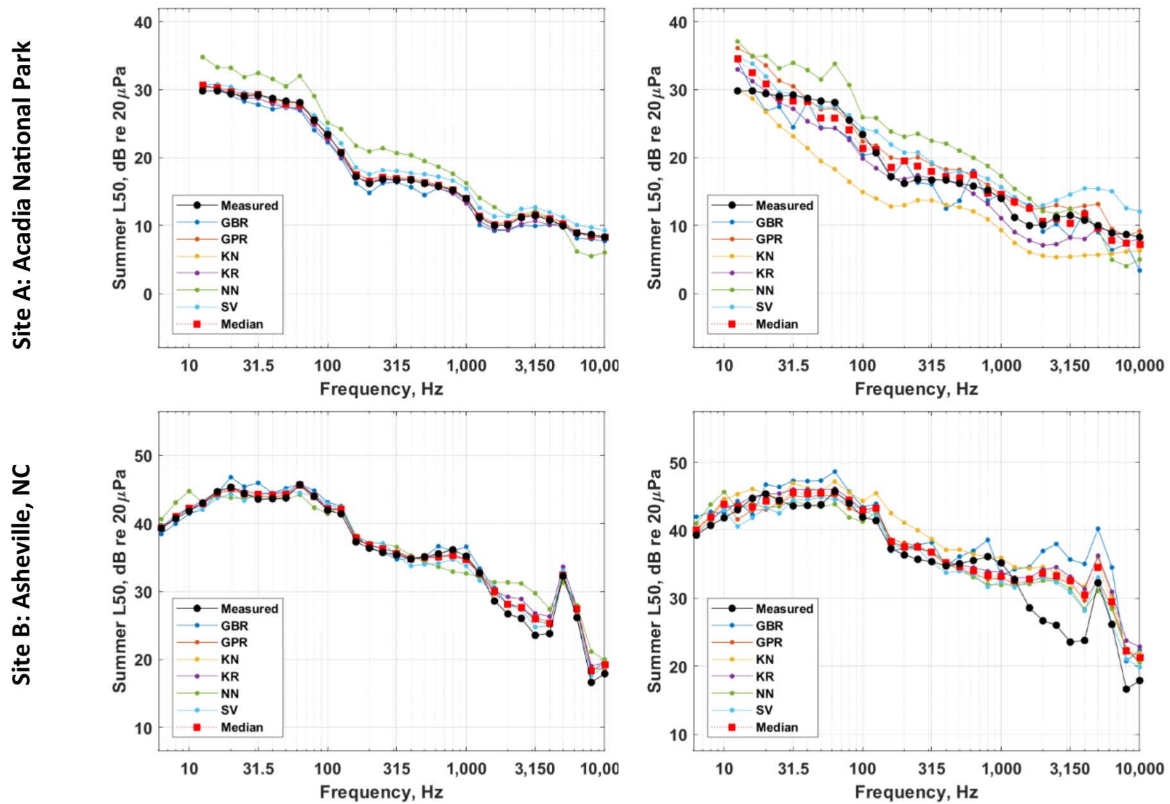

Figure 3.14 Predictions and measured levels at the first two of four chosen validation sites as a function of frequency. Plots on the left show nighttime model predictions when the complete training data set is used. Plots on the right are similar, but were created from predictions using a leave-one-out training data set. In other words, all models were trained using the complete training data set, excluding the site of interest.

www.manaraa.com

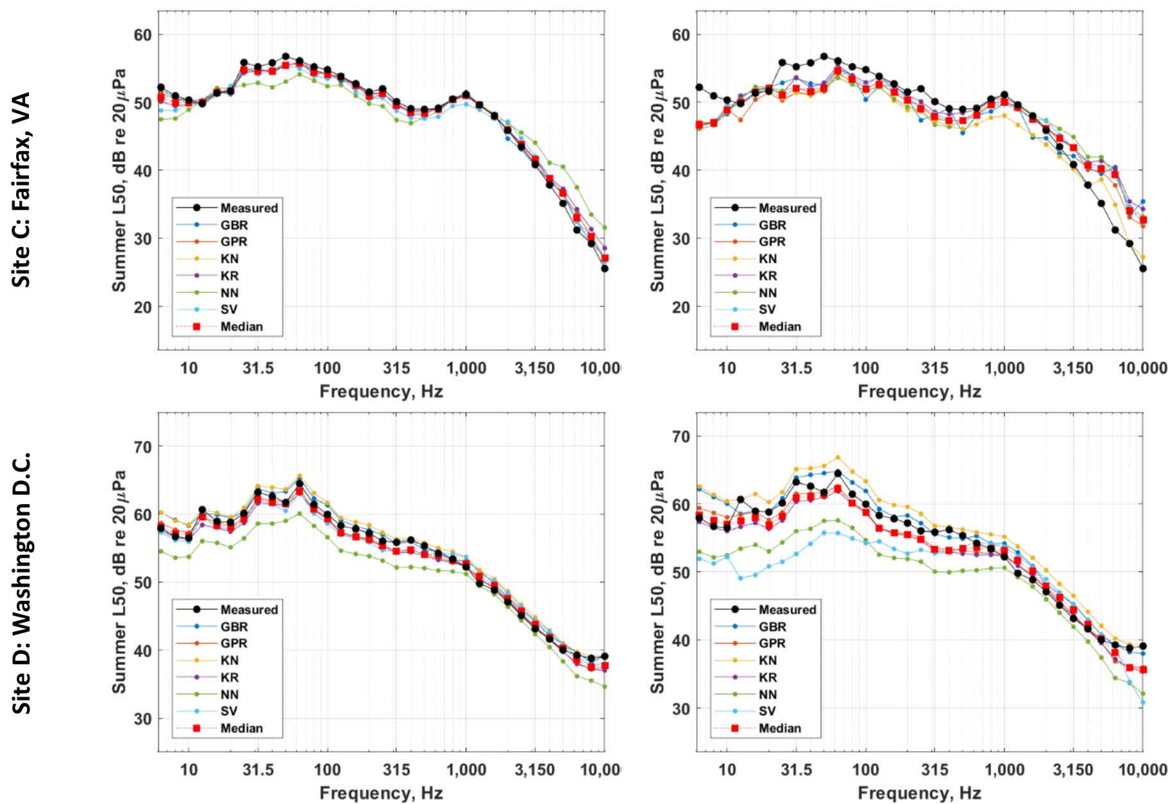

Figure 3.15 Predictions and measured levels at the last two chosen validation sites as a function of frequency. Plots on the left show nighttime model predictions when the complete training data set is used. Plots on the right are similar, but were created from predictions using a leave-one-out training data set. In other words, all models were trained using the complete training data set, excluding the site of interest.

المنارة للاستشارات
www.manaraa.com

models and the ensemble median, as well as the measured values at the four selected validation sites. These figures show the predicted values when all machine learning models are trained using the full training set (plots on the left) and when all machine learning models are trained using the full training set except for the site for which predictions were made (on the right). Note that leave-one-out predictions are not as close to the true measured values as full model predictions.

For all four sites, NNs are the only model that struggles to fit the measured data when the full training data set is used. This is likely because the NNs are very shallow to prevent overfitting of the training data. However, when we look at the leave-one-out predictions, the median tends to match the measured value fairly well. There are some exceptions to this. For example, none of the models were able to accurately predict the high frequency daytime levels in Acadia National Park or high frequency nighttime levels at the site in Asheville, NC. It is possible that the discrepancy in leave-one-out model predictions and measured values in Asheville and Acadia is due to the inability of the data to correctly characterize the amount of insect noise in a given location. Additionally, it is possible that the training data measurements do not reflect the large amounts of insect noise due to anomalies in insect activity during the time period when training data was collected. For all leave-one-out predictions, the median ensemble predictions are fairly stable and resilient to abrupt or drastic changes in a couple of the model predictions. For example, the KNN algorithm tends to predict values too low, but the NN tends to overpredict at Acadia National Park. The ensemble however does a fairly good job of matching the measured values.

Figures 3.16 and 3.17 show model predictions and measured values when all four validation sites are simultaneously removed from the training data set. Note that the leave-four-out and leave-one-out predictions are very similar. In both the leave-one-out and leave-four-out plots, the GBR model has a tendency to predict values which jump up and down as a function of frequency. This is more pronounced in the leave-four-out analysis, and could be due to the propensity of decision tree-type models to overfit the training data. The ensemble however does not have this

المنارة للاستشارات
www.manaraa.com

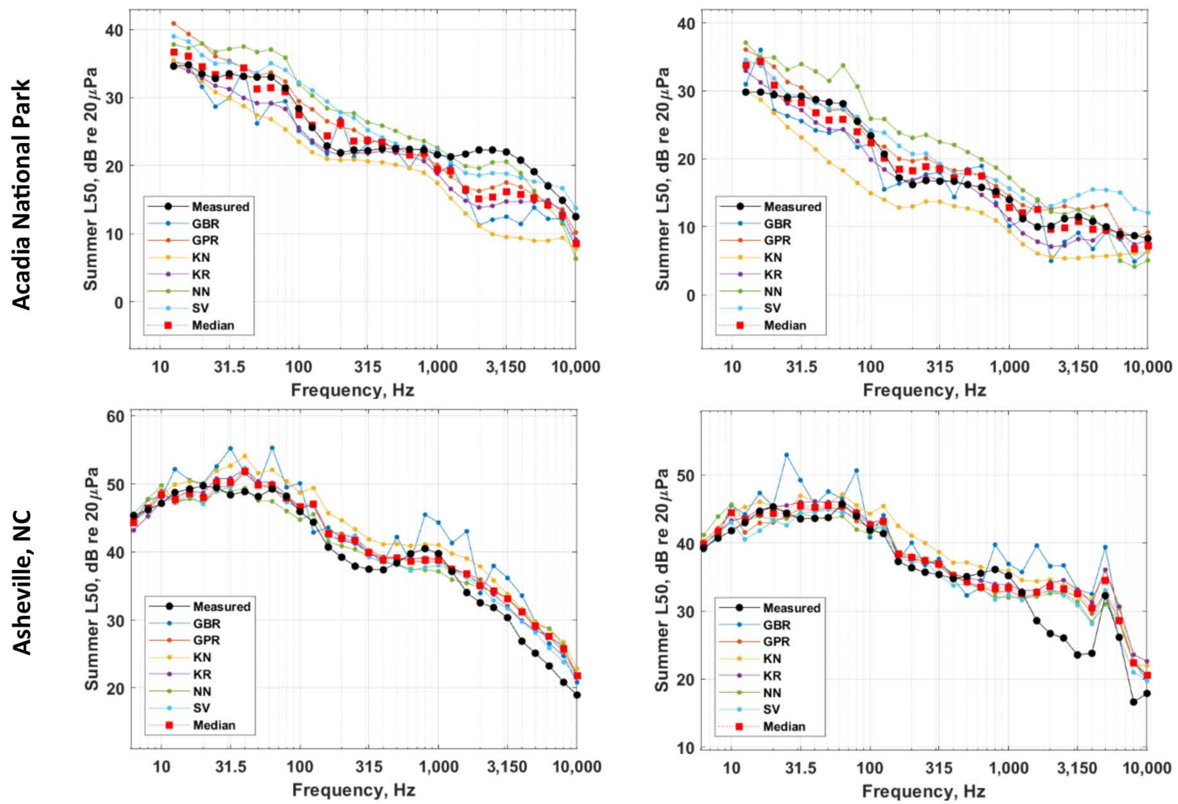

Figure 3.16 Plots on the left and right show daytime and nighttime model predictions respectively for the first two validation sites when the four validation sites were removed from the training data set. The plots show the predictions and measured levels at the four chosen validation sites as a function of frequency.

المنارة للاستشارات
www.manaraa.com

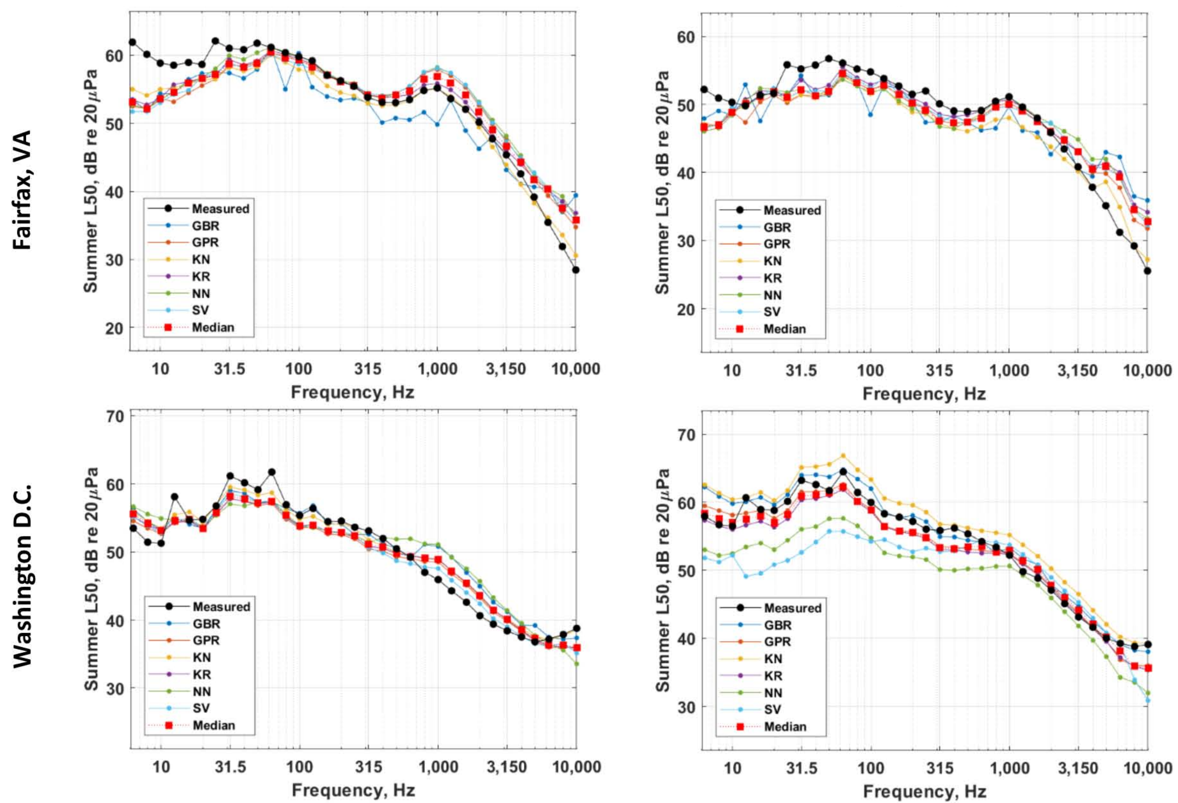

Figure 3.17 Plots on the left and right show daytime and nighttime model predictions respectively for the last two validation sites when the four validation sites were removed from the training data set. The plots show the predictions and measured levels at the four chosen validation sites as a function of frequency.

المنارة للاستشارات
www.manaraa.com

characteristic.

3.4 Feature Reduction

3.4.1 Initial Feature Importance Rankings

	Gini Importance	**Gini with Corr. Penalty**	**NN Weights**	**Human Intuition**
1	VIIRSMean_{270m}	VIIRSMean_{1080m}	Developed_{200m}	VIIRSMean_{270m}
2	Longitude	Elevation	Shrubland_{5000m}	DistStreamO4
3	Elevation	PPTWinter	RddMajor	Forest_{200m}
4	RddAll	DistAirpHeli	Elevation	DistRailroads
5	$\text{VIIRSMean}_{69120m}$	Longitude	$\text{Transportation}_{200m}$	DistAirpHeli
6	$\text{VIIRSMaximum}_{270m}$	DistMilitary	FlightFreq_{25km}	RddAll
7	VIIRSMean_{1080m}	PhysicalAccess	$\text{Institutional}_{5000m}$	TMaxSummer
8	DistAirpHeli	Latitude	WaterNat_{200m}	Cropland_{200m}
9	PPTWinter	DistStreamO1	UrbanHigh_{200m}	TdewAvgSummer
10	$\text{VIIRSMaximum}_{1080m}$	Slope	$\text{Commercial}_{5000m}$	DistStreamO3
11	DistMilitary	$\text{Extractive}_{5000m}$	$\text{VIIRSMaximum}_{270m}$	FlightFreq_{25km}
12	TMaxAnnual	DistRoadsMaj	Longitude	Cultivated_{200m}
13	DistRoadsMaj	Shrubland_{5000m}	Extractive_{200m}	DistMilitary
14	Slope	Suburban_{5000m}	VIIRSMean_{4320m}	DistRoadsAll
15	Forest_{5000m}	DistRailroads	Commercial_{200m}	PhysicalAccess

Table 3.2 The top 15 features found using the four metrics described in Sec. 2.4 for the summer daytime L_{50}. From left to right, these metrics are the Gini importance, Gini importance with a correlation penalty, a NN importance determined by weights, and human intuition. The features are ordered from most important at the top to less important at the bottom.

Table 3.2 shows the top 15 features as ranked by the four feature importance metrics discussed in Sec. 2.4 with lower numbered rankings corresponding to higher importance values. Note that these importance measures were made specifically for the summer daytime L_{50}. Also, to avoid overfitting the data, the NN model had no hidden layers. Hence, the importance calculated using NN weights only relies on the magnitude of the weight directly from a given feature to the output node. For an explanation of the features, refer to Appendix A.

المنارة للاستشارات
www.manaraa.com

It is interesting that three of the four metrics identified mean upward radiance at night as the most important feature. This suggests that anthropogenic noise is important to daytime L_{50} sound levels, as expected. The NN importance metric does not rank any upward radiance at night features higher than eleventh. However, it ranks the proportion of developed landcover as most important. This geospatial feature provides much of the same information as the upward radiance at night layers because they are both correlated with higher population densities and anthropogenic activity. So, all four metrics agree that anthropogenic sources are likely most important to determining the summer daytime L_{50} sound level.

All metrics, except for human intuition, rank longitude in the top 15 important features. Longitude provides no physical insight into what sound levels may be, but it does correlate with sound levels in our training data set. More specifically, most of the urban data are from the eastern CONUS area and most of the NPS data are from the western CONUS area. So, longitude affects sound level predictions, even though it has no physical significance. There are likely other features in these lists that happen to correlate with sound levels in the training data set, although they are not physically related to sound levels. It would be unwise to use these metrics to identify a reduced order model without first recognizing that these features are not necessarily the best choices for accurate model predictions.

The Gini importance was the only metric that did not identify any sources of water in the top 15 ranked features. Running water is a source of noise and water can also correlate with animal life and human habitation. The fact that the Gini importance did not identify sources of water as being very important is possibly due to sparsity of the data. Although some sites in the training data set are near water, there may not be enough for the GBR to recognize any connection between sound levels and water sources. Additionally, it is interesting to note that the Gini importance with a correlation penalty did conclude that the distance to streams is important.

Although these lists do not necessarily identify the top 15 features that affect the summer

المنارة للاستشارات
www.manaraa.com

daytime L_{50}, they do provide insight into possible issues with or characteristics of the training data and trained models. They allow us to get a general sense of whether or not the NN or GBR model is recognizing features that are physically important to sound propagation and absorption.

3.4.2 Error of Reduced Feature Model

Table 3.2 only shows the top 15 features because the leave-one-out MAD for the summer daytime L_{50} begins increasing when the number of features is reduced below about 15. Figures 3.18 and 3.19 show the leave-one-out MAD as a function of the number of features. Initially, I found the leave-one-out MAD using the original 117 features. Then, the feature with the lowest importance calculated using the Gini importance (Figure 3.18) or NN weights (Figure 3.19) was removed and the leave-one-out MAD was recalculated. This process was repeated until only one feature remained. The model hyperparameters of the individual models were not tuned during this process, but the model parameters were adjusted each iteration.

Both figures show the error is fairly constant when the number of features is greater than about 15. The NN model does perform worse than most models for a smaller number of features, particularly when using the Gini importance. However, tuning the model hyperparameters would likely prevent the NN models from struggling as much. Even though the NN models have higher errors when fewer features are used, the median error is fairly constant down to about 15 features. Although not shown here, the leave-one-out RMSE shows similar behavior with feature reduction.

3.4.3 Changes in Predictions from Reduced Feature Models

Four reduced feature models were trained using the 15 features in each column of Table 3.2. Maps were created to show the difference in model predictions between the new model predictions from the reduced feature data sets and predictions made with the full data set of 117 features. Figures 3.20 and 3.21 show the change in sound level predictions for the summer daytime L_{50}

المنارة للاستشارات
www.manaraa.com

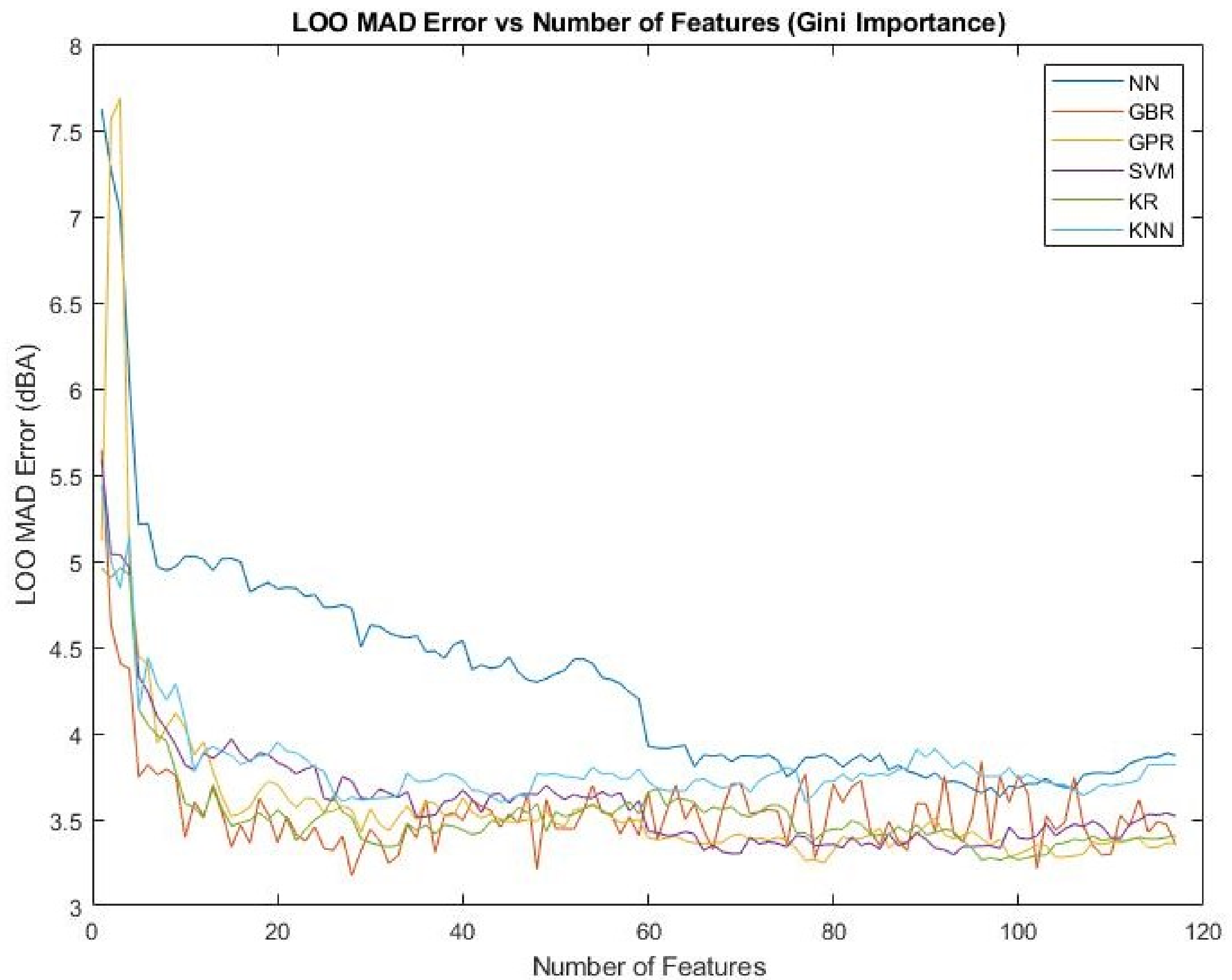

Figure 3.18 Plot of the leave-one-out (LOO) MAD for the summer daytime L_{50} for all models in the ensemble as the number of features is reduced from the original 117. The feature corresponding to the smallest Gini importance was removed each iteration.

المنارة للاستشارات
www.manaraa.com

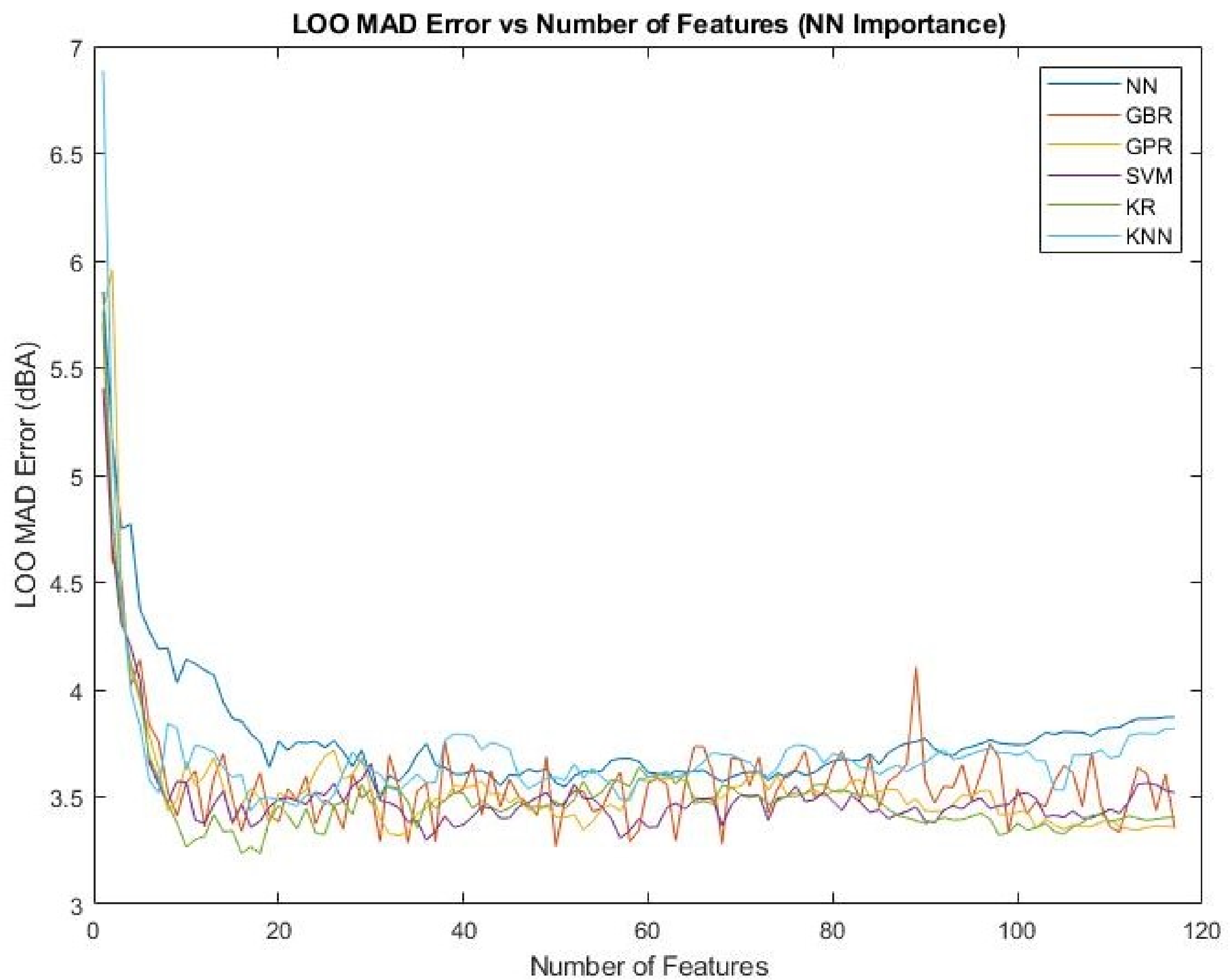

Figure 3.19 Plot of the leave-one-out (LOO) MAD for the summer daytime L_{50} for all models in the ensemble as the number of features is reduced from the original 117. The feature corresponding to the smallest importance as measured using NN weights was removed each iteration.

المنارة للاستشارات
www.manaraa.com

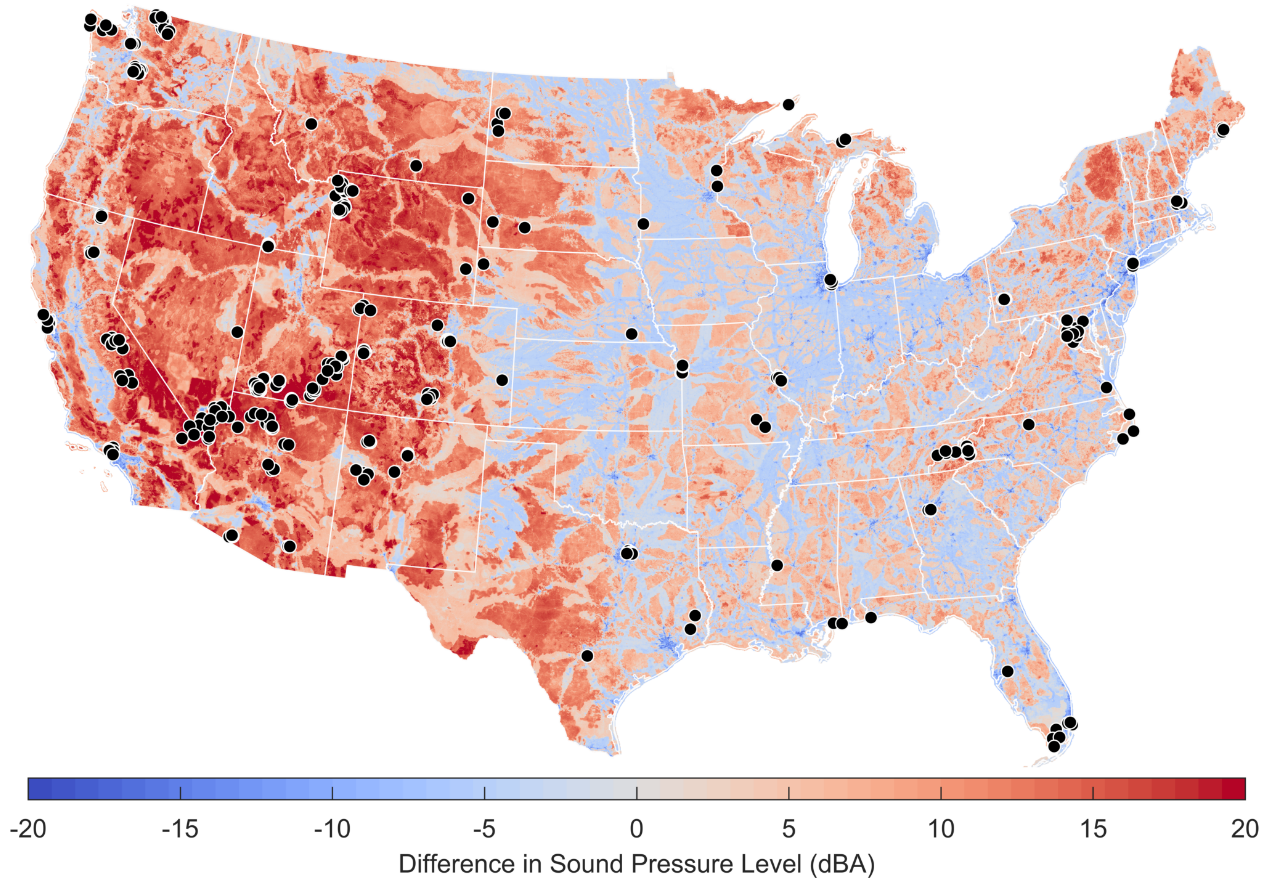

Figure 3.20 Map of the difference between summer daytime L_{50} ensemble predictions using the top 15 features identified by the Gini importance with a correlation penalty and the full model with 117 features. Blue areas correspond to a drop in sound level predictions from the reduced feature model when compared to the full model. Similarly, red areas correspond to an increase in sound level predictions from the reduced feature model.

using the top 15 features identified by the Gini importance with a correlation penalty and human intuition respectively. Blue (red) areas represent locations where the levels decreased (increased) using the reduced model instead of the full model. Figure 3.20 shows greater differences, especially where levels have increased (red areas), than Figure 3.21. The scale on these maps is from -20 to $+20$ dBA, so the differences shown here, especially in Figure 3.20, are significant. Although differences shown in Figure 3.21 are not as large, they are still large in several areas, particularly in the western CONUS area.

The fact that the leave-one-out MAD is approximately the same for the reduced models, and yet

المنارة للاستشارات
www.manaraa.com

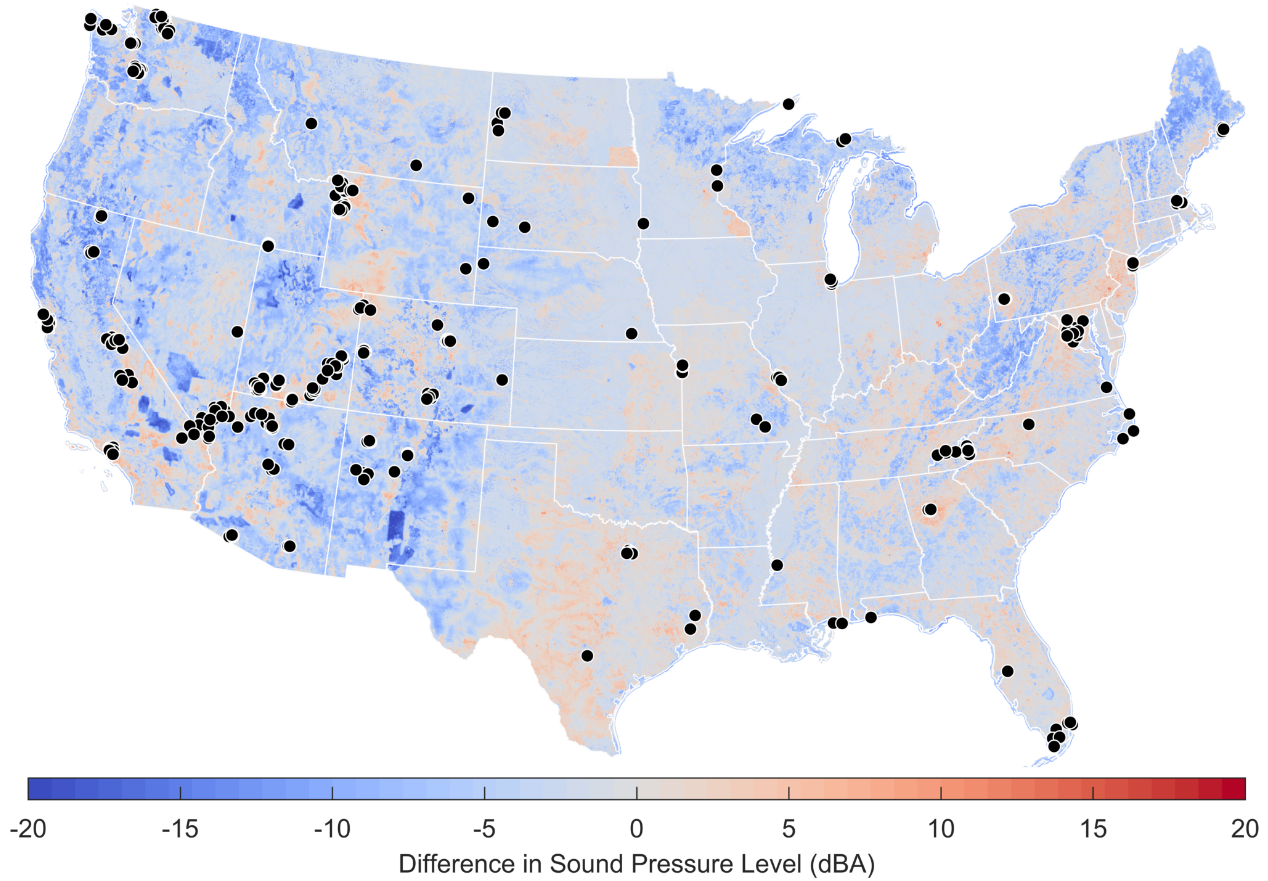

Figure 3.21 Map of the difference between summer daytime L_{50} ensemble predictions using the top 15 features identified by human intuition and the full model with 117 features. Blue areas correspond to a drop in sound level predictions from the reduced feature model when compared to the full model. Similarly, red areas correspond to an increase in sound level predictions from the reduced feature model.

المنارة للاستشارات
www.manaraa.com

the predictions are very different, shows how limited the data set is. Additionally, it is important to select features that are physically meaningful and hold explanatory power, rather than just selecting the features that are identified by various measures of feature importance.

المنارة للاستشارات
www.manaraa.com

Chapter 4

Conclusions and Future Work

4.1 Conclusion

I have successfully created a computational pipeline that predicts ambient sound levels using an ensemble determined by the median prediction of six machine learning algorithms, GBR, NNs, KNN, SVMs, KR, and GPR. Predictions can be made for a variety of acoustic metrics, geographic areas, frequency bands, and timeframes. I performed preliminary validation studies on the ensemble by finding the standard deviation of the ensemble predictions, which is representative of the structural uncertainty. Standard deviation values identified locations of greater uncertainty, which are good candidates for data collection. Best practices for improving machine learning models in the limited data regime, and measuring the performance of such models have still not been determined. However, this research has made steps towards identifying such methods.

4.2 Future Work

Although unsupervised learning was not used, several unsupervised techniques are adept at identifying patterns or structures in a data set. In particular, unsupervised learning methods may

للاستشارات المنارة
www.manaraa.com

help identify regions that are void from or underrepresented in the training data set. Sites in these regions would be good candidates for data collection, particularly if ensemble standard deviation values are also high.

In addition to collecting more acoustic training data, I could search for additional geospatial databases, which may provide novel information to the model. For example, if I could find a geospatial feature correlated with insect activity or populations, I could potentially improve our predictions of high frequency noise. There may be other geospatial features that are more useful than the ones in the data set currently. It is also worth considering incorporating known physics-based models, such as road noise models, into the data set.

All maps shown here, except those in Sec. 3.4.3, were generated using all available geospatial features as inputs. Further work could be done in identifying reduced models that utilize geospatial features with high explanatory power. Reduced parameter models would be less computationally expensive and possibly aid in preventing overfitting.

المنارة للاستشارات
www.manaraa.com

Appendix A

Geospatial Features

Table A.1 CONUS geospatial features, their area of analysis, description, and units.

Variable	Area of Analysis	Description	Units
Elevation	Point	Digital elevation, height above sea level	m
Slope	Point	Rate of change of elevation	Degrees
PPTSummer	Point	10-year average summer precipitation	mm
PPTWinter	Point	10-year average winter precipitation	mm
PPTAnnual	Point	10-year average yearly precipitation	mm
TMaxSummer	Point	10-year average summer maximum temperature	°C
TMaxWinter	Point	10-year average winter maximum temperature	°C
TMaxAnnual	Point	10-year average yearly maximum temperature	°C
TMinSummer	Point	10-year average summer minimum temperature	°C
TMinWinter	Point	10-year average winter minimum temperature	°C
TMinAnnual	Point	10-year average yearly minimum temperature	°C
TdewAvgSummer	Point	10-year average summer minimum dew point	°C

Continued on next page

للاستشارات

www.manaraa.com

Table A.1 – *Continued from previous page*

Variable	Area of Analysis	Description	Units
TdewAvgWinter	Point	10-year average winter maximum dew point	°C
TdewAvgAnnual	Point	10-year average yearly minimum dew point	°C
Barren	200 m, 5 km	Proportion of barren landcover	%
Cultivated	200 m, 5 km	Proportion of cultivated landcover	%
Deciduous	200 m, 5 km	Proportion of deciduous forest landcover (level 2)	%
Developed	200 m, 5 km	Proportion of developed landcover	%
Evergreen	200 m, 5 km	Proportion of evergreen forest landcover (level 2)	%
Forest	200 m, 5 km	Proportion of forest landcover	%
Herbaceous	200 m, 5 km	Proportion of herbaceous landcover	%
MixedForest	200 m, 5 km	Proportion of mixed forest landcover (level 2)	%
Shrub	200 m, 5 km	Proportion of shrubland landcover	%
Water	200 m, 5 km	Proportion of water (only) landcover	%
Wetland	200 m, 5 km	Proportion of wetlands landcover	%
DistCoast	Point	Distance to nearest coastline	m
DistStreamO	Point	Distance to nearest stream with Strahler order greater than 1, 3, or 4	m
DistWaterBody	Point	Distance to nearest body of water	m
Built	200 m, 5 km	Degree of human modification from built land use	Ratio
Commercial	200 m, 5 km	Degree of human modification from commercial land use	Ratio
Cropland	200 m, 5 km	Degree of human modification from cropland land use	Ratio
DistAirpHeli	Point	Distance to nearest heliport	m
DistAirpHigh	Point	Distance to nearest high-volume airport	m
DistAirpLow	Point	Distance to nearest low-volume airport	m
DistAirpMod	Point	Distance to nearest moderate-volume airport	m

Continued on next page

للاستشارات المنارة
www.manaraa.com

Table A.1 – *Continued from previous page*

Variable	Area of Analysis	Description	Units
DistAirpMoto	Point	Distance to nearest motorized airport	m
DistAirpSea	Point	Distance to nearest seaplane airport	m
DistMilitary	Point	Distance to nearest military flight path	m
DistRailroads	Point	Distance to nearest rail line	m
DistRoadsAll	Point	Distance to nearest road (all roads)	m
DistRoadsMaj	Point	Distance to nearest road (major roads)	m
Extractive	200 m, 5 km	Degree of human modification from extractive land use	Ratio
ExurbanHigh	200 m, 5 km	Degree of human modification from high exurban land use	Ratio
ExurbanLow	200 m, 5 km	Degree of human modification from low exurban land use	Ratio
FlightFreq	25 km	Total weekly flight observations	Count
Grazing	200 m, 5 km	Degree of human modification from grazing land use	Ratio
Industrial	200 m, 5 km	Degree of human modification from industrial land use	Ratio
Institutional	200 m, 5 km	Degree of human modification from institutional land use	Ratio
MilitarySum	40 km	Sum of designated military flight paths	Count
Mining	200 m, 5 km	Degree of human modification from mining land use	Ratio
Park	200 m, 5 km	Degree of human modification from park land use	Ratio
Pasture	200 m, 5 km	Degree of human modification from pasture land use	Ratio
PhysicalAccess	Point	Travel time given transportation infrastructure and off-trail permeability	Ratio
RddAll	Point, 5 km	Road density, sum of road lengths (all roads) divided by area of interest	km/km^2
RddMajor	Point, 5 km	Road density, sum of road lengths (major roads only) divided by area of interest	km/km^2

Continued on next page

المنارة للاستشارات
www.manaraa.com

Table A.1 – *Continued from previous page*

Variable	Area of Analysis	Description	Units
RecCon	200 m, 5 km	Degree of human modification from recreation-conservation land use	Ratio
Suburban	200 m, 5 km	Degree of human modification from suburban land use	Ratio
Timber	200 m, 5 km	Degree of human modification from timber land use	Ratio
Transportation	200 m, 5 km	Degree of human modification from transportation land use	Ratio
UrbanHigh	200 m, 5 km	Degree of human modification from high urban land use	Ratio
UrbanLow	200 m, 5 km	Degree of human modification from low urban land use	Ratio
VIIRS	270 m, 1080 m, 4320 m, 17280 m, 69120 m	Maximum, mean, and minimum upward radiance at night	$nW/cm^2/sr$
WaterHum	200 m, 5 km	Degree of human modification from water land use	Ratio
WaterNat	200 m, 5 km	Degree of human modification from natural water land use	Ratio
Wet	200 m, 5 km	Degree of human modification from wet land use	Ratio
Latitude	Point	Latitude value of raster cell in decimal degrees	Degrees
Longitude	Point	Longitude value of raster cell in decimal degrees	Degrees

المنارة للاستشارات
www.manaraa.com

Appendix B

Acoustic Data

Table B.1 All A and flat-weighted summertime acoustic metrics and the number of measurements for each metric from the NPS, BRRC, the EPA, and a trusted third-party consulting firm.

Summer Acoustic Metric	NPS	BRRC	EPA	Consulting Firm
L_1 dBA day	326	54	0	22
L_1 dBA hour	326	54	0	22
L_1 dBA night	326	54	0	22
L_1 dBZ day 1/3 octave bands	0	54	0	22
L_1 dBZ day frequency groups	0	54	0	22
L_1 dBZ hour 1/3 octave bands	0	54	0	22
L_1 dBZ hour frequency groups	0	54	0	22
L_1 dBZ night 1/3 octave bands	0	54	0	22
L_1 dBZ night frequency groups	0	54	0	22
L_5 dBZ hour 1/3 octave groups	326	0	0	0
L_5 dBZ hour frequency groups	326	0	0	0
L_{10} dBA day	326	54	100	22
L_{10} dBA hour	326	54	100	22
L_{10} dBA night	326	54	100	22
L_{10} dBZ day 1/3 octave bands	309	54	0	22
L_{10} dBZ day frequency groups	309	54	0	22
L_{10} dBZ hour 1/3 octave bands	0	54	0	22
L_{10} dBZ hour frequency groups	0	54	0	22
L_{10} dBZ night 1/3 octave bands	308	54	0	22
L_{10} dBZ night frequency groups	308	54	0	22
L_{50} dBA day	326	54	100	22
L_{50} dBA hour	326	54	100	22
L_{50} dBA night	326	54	100	22

Continued on next page

المنارة للاستشارات
www.manaraa.com

Table B.1 – *Continued from previous page*

Summer Acoustic Metric	**NPS**	**BRRC**	**EPA**	**Consulting Firm**
L_{50} dBZ day 1/3 octave bands	309	54	0	22
L_{50} dBZ day frequency groups	309	54	0	22
L_{50} dBZ hour 1/3 octave bands	326	54	0	22
L_{50} dBZ hour frequency groups	326	54	0	22
L_{50} dBZ night 1/3 octave bands	308	54	0	22
L_{50} dBZ night frequency groups	308	54	0	22
L_{90} dBA day	326	54	100	22
L_{90} dBA hour	326	54	100	22
L_{90} dBA night	326	54	100	22
L_{90} dBZ day 1/3 octave bands	309	54	0	22
L_{90} dBZ day frequency groups	309	54	0	22
L_{90} dBZ hour 1/3 octave bands	326	54	0	22
L_{90} dBZ hour frequency groups	326	54	0	22
L_{90} dBZ night 1/3 octave bands	308	54	0	22
L_{90} dBZ night frequency groups	308	54	0	22
L_{99} dBA day	326	54	0	22
L_{99} dBA hour	326	54	0	22
L_{99} dBA night	326	54	0	22
L_{99} dBZ day 1/3 octave bands	0	54	0	22
L_{99} dBZ day frequency groups	0	54	0	22
L_{99} dBZ hour 1/3 octave bands	0	54	0	22
L_{99} dBZ hour frequency groups	0	54	0	22
L_{99} dBZ night 1/3 octave bands	0	54	0	22
L_{99} dBZ night frequency groups	0	54	0	22
L_{eq} dBA day	326	0	0	0
L_{eq} dBA hour	326	0	0	0
L_{eq} dBA night	326	0	0	0
L_{max} dBA day	326	0	0	0
L_{max} dBA hour	326	0	0	0
L_{max} dBA night	326	0	0	0
L_{min} dBA day	326	0	0	0
L_{min} dBA hour	326	0	0	0
L_{min} dBA night	326	0	0	0

www.manaraa.com

Appendix C

Pipeline Code

C.1 README

Instructions for making maps:

Part 1:

1. ParseData: Select the correct training data name and input data name to control which data versions are used for training and making predictions. Change the output name to match these. (The most recent data version was 20180315, and the one before was 20170925.)

2. Main.py: This is the actual script you will run. Set the regions, modelNames, and corresponding models you want to run. Also select the Data files you want to run.

3. When ParseData and Main are ready, type:
nohup /home/mark/data/anaconda/envs/python3/bin/python3 Main.py &. This will start the program running in the background, and it will continue running even if you close your current terminal. All output is written to a file called nohup.out. The long file path is required to use the correct version of Python. The program will typically take 5.5 hours for all of CONUS and a single acoustic metric.

4. Wait for it to run. The output will be sent to nohup.out, so you can check on the program output there.

Part 2:

1. You now have NW (northwest), NE (northeast), SW (southwest), and SE (southeast) files, but want them to be combined into one CONUS file. Edit createFullConus to have the correct folderPath, data, and metric.

2. Run: /home/mark/data/anaconda/envs/python3/bin/python3 createFullConus.py

3. Now, you want to create the ensemble (median) and standard deviation files. Edit findMedian to have the correct folderPath, data, and metric again.

المنارة للاستشارات
www.manaraa.com

4. Run: /home/mark/data/anaconda/envs/python3/bin/python3 findMedian.py

5. Delete the region prediction files that you created, since you now have the complete CONUS map and they are no longer needed.

C.2 Main.py

```
# Main
import time
from PipelinesPool import *
from ParseData import *
from PrunedModels import *
import numpy as np

import sys
sys.path.append('../')
sys.path.append('DataFiles/')

# Models I want to run:
# All 6 pruned models: GPR_pruned, KN_pruned, KR_pruned, NN_pruned, GBR_pruned,
    SV_pruned
# Maps to create (data files, all of these have Std Scaler):
# Data1005-Data1010: L10, 50, 90 D/N
# Data1017-Data1040: L50 hourly
# Data1065-Data1100: L50 frequency bands
# Data1193-Data1198: L50 fgroups D/N

regions=['NW','SW','NE','SE']

modelNames=[GPR_pruned,KN_pruned,KR_pruned,GBR_pruned,NN_pruned,SV_pruned]

modNames=["GPR_pruned","KN_pruned","KR_pruned","GBR_pruned","NN_pruned","SV_pruned"
    ]

import Data1005
import Data1006
import Data1007
import Data1008
import Data1009
import Data1010

dataset=[Data1005,Data1006,Data1007,Data1008,Data1009,Data1010]
numCores=45
```

المنارة للاستشارات
www.manaraa.com

```
for data in dataset:
        for region in regions:
                start=time.time()
                MapMasterFunction(data.data_name+'_'+region+'_', modNames,
                    modelNames, data.X, data.Xscaled, data.Xscaler, data.Y, data.
                    Yscaled, data.Yscaler, data.features,
                        data.layers, data.soundFormats, region, numCores, ParseData.
                            output_folder, True, False, False, False, 20)
                end=time.time()
                print('Total time for '+data.data_name+' and %s: %s' % (region,end-
                    start))
```

C.3 ParseData.py

```
import numpy as np
import scipy.io
from os import listdir
import os
import shutil
import math

data_version = "20180315"

#data_name = "20180315_CONUS_TRAINING"
data_name = "20170925_CONUS_TRAINING"

#input_name = data_version+'_NC_INPUT'
input_name = data_version + '_CONUS_INPUT'

output_name = '20170925_CONUS_OUTPUT'

output_folder = "../results/%s/" % output_name

os.chdir('../results/')
if not os.path.exists(output_name):
    try:
        os.mkdir(output_name)
    except OSError as exc: # Guard against race condition
        print("Error creating output folder.")
os.chdir('../src/')

# Load matlab training data
data = scipy.io.loadmat("../data/%s/%s.mat" % (data_name, data_name))
```

المنارة للاستشارات
www.manaraa.com

```
sound = data["SOUND"][0]
acoustic_variable_index = [descr[0] for descr in sound.dtype.descr].index(
    "Variable") # Different data sets put the description index in different
        columns

N = len(sound)

# Load Geospatial layers
geospatial = data["GEOSPATIAL"]

M = len(geospatial[0]["v"])
numSites = len(geospatial[0, 0]["v"])

# Extracting all geospatial feature values and feature names:
geo_data = np.zeros([numSites, M])
for i in range(M):
    geo_data[:, i] = geospatial[0, i]["v"][:, 0]

geo_names = []
for i in range(M):
    geo_names.append(geospatial[0, i]['Variable'][0])


def LoadInputData(input_name,features,region, numpy_Bool=False):
    # load matlab/numpy input data
    geoData=0
    rows=0
    cols=0

    if numpy_Bool==True:
        dataIn=np.load('../data/%s/GEOSPATIAL_ALL_FEATURES_%s.npy' % (input_name,
            region))
        featIn=open('../data/%s/GEOSPATIAL_ALL_FEATURES_%s.txt' % (input_name,
            region))
        featList=featIn.read().split('\n')
        featList.remove('')
        rows, cols=np.shape(dataIn[0])
        geoData=np.zeros((rows*cols,len(features)))
        for j in range(len(features)):
            geoData[:,j]=dataIn[featList.index(features[j]),:,:].flatten()
    else:
        dataIn=scipy.io.loadmat('../data/%s/GEOSPATIAL_ALL_FEATURES_%s.mat' % (
            input_name, region))
        #print(dataIn.keys())
```

www.manaraa.com

```
        rows,cols=np.shape(dataIn[features[0]])
        geoData=np.zeros((rows*cols,len(features)))
        for j in range(len(features)):
            geoData[:,j]=dataIn[features[j]].flatten()

    mapDim=[rows,cols]
    print(mapDim)
    return geoData, mapDim

def get_GeospatialNames():
    """
    Returns a list of geospatial names in the current data set
    """
    return geo_names

def LoadData(layers=["Summer_L50_dBA_day", "Summer_L10_dBA_day", "
    Summer_L90_dBA_day", "Summer_L50_dBA_ngt",
                    "Summer_L10_dBA_ngt", "Summer_L90_dBA_ngt", ]):
    """
    Load the acoustical data corresponding to the names in layers
    """
    acoustic_data = np.zeros((numSites,len(layers)))
    allMetrics = [sound[i][1][0] for i in range(N)]
    soundFormats=[]
    for i in range(len(layers)):
        if ('fgroups' in layers[i] and 'hr' not in layers[i]):
            name, col = layers[i].split('fgroups',1)
            name = name +'fgroups'
            col=int(col)-1
            index=allMetrics.index(name)

            copyFormat=sound[index].copy()
            copyFormat['freq']=copyFormat['freq'][:,col].reshape(2,1)
            soundFormats.append(copyFormat)

            acoustic_data[:,i]=sound[index][0][:,col]

        elif ('fgroups' in layers[i] and 'hr' in layers[i]):
            split1,split2=layers[i].split('_hr',1)
            hr,group=split2.split('_fgroups')
            hrNum=int(hr)-1
            groupNum=int(group)-1
            name=split1+'_hr_fgroups'
            index=allMetrics.index(name)
```

المنارة للاستشارات
www.manaraa.com

```
        copyFormat=sound[index].copy()
        copyFormat['freq']=copyFormat['freq'][:,groupNum].reshape(2,1)
        copyFormat['hrs']=np.array([[hrNum]],dtype=np.uint8)
        soundFormats.append(copyFormat)

        acoustic_data[:,i]=sound[index][0][:,hrNum,groupNum]

    elif ('day_f' in layers[i] or 'ngt_f' in layers[i]):
        split1,split2=layers[i].split('_f',1)
        fNum=int(split2)-1
        name=split1+'_f'
        index=allMetrics.index(name)

        copyFormat=sound[index].copy()
        copyFormat['freq']=np.array([[copyFormat['freq'][0,fNum]]])
        soundFormats.append(copyFormat)

        acoustic_data[:,i]=sound[index][0][:,fNum]

    elif('dBZ_hr' in layers[i]):
        split1,split2=layers[i].split('_hr')
        hr,f=split2.split('_f')
        hrNum=int(hr)-1
        fNum=int(f)-1
        name=split1+'_hr_f'
        index=allMetrics.index(name)

        copyFormat=sound[index].copy()
        copyFormat['freq']=np.array([[copyFormat['freq'][0,fNum]]])
        copyFormat['hrs']=np.array([[hrNum]],dtype=np.uint8)
        soundFormats.append(copyFormat)

        acoustic_data[:,i]=sound[index][0][:,hrNum,fNum]

    elif('dBA_hr' in layers[i]):
        split1,hr=layers[i].split('_hr')
        hrNum=int(hr)-1
        name=split1+'_hr'
        index=allMetrics.index(name)

        copyFormat=sound[index].copy()
        copyFormat['hrs']=np.array([[hrNum]],dtype=np.uint8)
        soundFormats.append(copyFormat)
```

www.manaraa.com

```
            acoustic_data[:,i]=sound[index][0][:,hrNum]

        else:
            index=allMetrics.index(layers[i])

            copyFormat=sound[index].copy()
            soundFormats.append(copyFormat)

            acoustic_data[:,i]=sound[index][0].flatten()

    if len(layers) == 1:
        return geo_data, acoustic_data.flatten(), soundFormats
    else:
        return geo_data, acoustic_data, soundFormats

def RemoveAllNans(geo_data, acoustic_data):
    """
    Remove any sites that have only nan values in their acoustical data
    """
    geoNanMatrix=np.isnan(geo_data)
    for i in range(np.shape(geoNanMatrix)[1]):
        if True in geoNanMatrix[:,i]:
            print("There␣are␣NaN␣values␣in␣the␣geo_data␣in␣column␣" + str(i+1))

    m = acoustic_data.shape[0]
    dim = acoustic_data.ndim
    # remove training sites with no acoustic data
    ind_to_keep=[]
    nanMatrix=np.isnan(acoustic_data)
    for i in range(m):
        if dim==1:
            if True == nanMatrix[i]:
                continue
            else:
                ind_to_keep.append(i)
        else:
            if False in nanMatrix[i,:]:
                ind_to_keep.append(i)
            else:
                continue

    return geo_data[ind_to_keep], acoustic_data[ind_to_keep]
```

www.manaraa.com

```
def RemoveAnyNans(geo_data, acoustic_data):
    """
    Remove any sites that have any nan values in their acoustical data
    """
    geoNanMatrix=np.isnan(geo_data)
    for i in range(np.shape(geoNanMatrix)[1]):
        if True in geoNanMatrix[:,i]:
            print("There␣are␣NaN␣values␣in␣the␣geo_data␣in␣column␣" + str(i+1))

    m = acoustic_data.shape[0]
    dim = acoustic_data.ndim
    # remove training sites with no acoustic data
    ind_to_keep=[]
    nanMatrix=np.isnan(acoustic_data)
    for i in range(m):
        if dim==1:
            if True == nanMatrix[i]:
                continue
            else:
                ind_to_keep.append(i)
        else:
            if True in nanMatrix[i,:]:
                continue
            else:
                ind_to_keep.append(i)

    return geo_data[ind_to_keep], acoustic_data[ind_to_keep]

def RemoveCOData(geo_data, acoustic_data, feature_names):
    """
    Removing the Colorado sites along the river that are really loud.
    """
    latIndex = feature_names.index('Latitude')
    lonIndex = feature_names.index('Longitude')
    distToStreamIndex = feature_names.index('DistStream01')

    numSites = geo_data.shape[0]

    pruned_inds = []
    for i in range(numSites):
        if (36 < geo_data[i, latIndex] < 36.37 and -113.4 < geo_data[i, lonIndex] <
             -112.2 and geo_data[
            i, distToStreamIndex] < 200):
```

المنارة للاستشارات
www.manaraa.com

```
            continue
        else:
            pruned_inds.append(i)

    return geo_data[pruned_inds], acoustic_data[pruned_inds]


def RemoveListOfFeatures(geo_data, geo_names, geo_names_to_remove):
    """
    Removing the features provideed in the geo_names_to_remove list.
    """
    ind_to_keep = []
    for name in geo_names:
        if name in geo_names_to_remove:
            continue
        else:
            ind_to_keep.append(geo_names.index(name))

    for name in geo_names_to_remove:
        if name not in geo_names:
            print(name+"␣is␣not␣a␣feature␣name.")

    geo_names = [i for i in geo_names if i not in geo_names_to_remove]

    return geo_data[:, ind_to_keep], geo_names


def KeepListOfFeatures(geo_data, geo_names, geo_names_to_keep):
    """
    Keeping the features provideed in the geo_names_to_keep list.
    """
    ind_to_keep = []
    for name in geo_names:
        if name in geo_names_to_keep:
            ind_to_keep.append(geo_names.index(name))

    for name in geo_names_to_keep:
        if name not in geo_names:
            print(name+ "␣is␣not␣a␣feature␣name.")
    return geo_data[:, ind_to_keep], geo_names_to_keep
```

C.4 PipelinesPool.py

```
import Validation
```

المنارة للاستشارات
www.manaraa.com

```
from sklearn.multioutput import MultiOutputRegressor
import Plotting
import pylab
import scipy.io as sio
import numpy as np
import logging
import time
import math
from AnalysisTools import *
from Scalers import *
import ParseData
from multiprocessing import Pool
import multiprocessing

def GetModel(single_output_model, Y, n_jobs=1):
    if len(Y.shape) == 2:
        return MultiOutputRegressor(single_output_model, n_jobs=n_jobs)
    else:
        return single_output_model


def Fit(name, single_output_model, X, Xscaled, Xscaler, Y, Yscaled, Yscaler,
    features, layers, n_jobs=1, results_path=ParseData.output_folder, rfe=False,
    num_features=20):
    # Recursive feature elimination can be used.
    model = GetModel(single_output_model, Yscaled, n_jobs)

    logging.info("Performing␣Fit␣for␣model␣%s" % name)

    if rfe==True:
        rfe, model, features = TryRFE(model, Xscaled, Yscaled, features,
            num_features)
    else:
        model.fit(Xscaled, Yscaled)

    fit_predicted_scaled = model.predict(Xscaled)
    fit_predicted = Yscaler.inverse_transform(fit_predicted_scaled)

    # If possible, create a global variable importance list:
    GlobalVariableImportanceMeasures(name, model, features, layers, results_path,
        rfe, num_features)

    logging.info("Fitting␣Results:␣RMSE␣=␣%02f,␣MAD␣=␣%02f" % (Validation.RMSE(Y,
        fit_predicted), Validation.MAD(Y, fit_predicted)))
```

المنارة للاستشارات
www.manaraa.com

```
    Plotting.PlotErrors(Y.flatten(), fit_predicted.flatten(), "%s␣Fitting␣Error" %
        name)
    pylab.savefig("%s%s_fit_error.png" % (results_path, name), dpi=300)
    pylab.close()
    return model, fit_predicted, fit_predicted_scaled


def LOOCV(name, single_output_model, X, Xscaled, Xscaler, Y, Yscaled, Yscaler,
    n_jobs=1, results_path=ParseData.output_folder, rfe=False, num_features=20):
    # If recursive feature elimination is used, if will only be performed once in
        Validation.LOOCV.
    model = GetModel(single_output_model, Yscaled, n_jobs)

    # Cross Validate
    logging.info("Performing␣LOOCV␣on␣model␣%s" % name)

    loocv_predicted_scaled = Validation.LOOCV(model, Xscaled, Yscaled, rfe,
        num_features)
    loocv_predicted = Yscaler.inverse_transform(loocv_predicted_scaled)

    # Print some results
    logging.info("LOOCV␣Results:␣RMSE␣=␣%02f,␣MAD␣=␣%02f" % (Validation.RMSE(Y,
        loocv_predicted), Validation.MAD(Y, loocv_predicted)))

    Plotting.PlotErrors(Y.flatten(), loocv_predicted.flatten(), "%s␣LOOCV␣Error" %
        name)
    pylab.savefig("%s%s_loocv_error.png" % (results_path, name), dpi=300)
    pylab.close()
    return loocv_predicted, loocv_predicted_scaled


def MapMasterFunction(name, modelNames, models, X, Xscaled, Xscaler, Y, Yscaled,
    Yscaler, features, layers, soundFormats, region, num_jobs_creating_map,
    results_path, createMap, anthropogenicMap, naturalMap, rfe, num_features):
    #pool=Pool(processes=num_jobs_creating_map)
    soundmap=np.array([])
    natMap=np.array([])
    anthMap=np.array([])
    numOutputs=len(layers)

    loadStart=time.time()
    mapinput,mapDim=ParseData.LoadInputData(ParseData.input_name,features,region)
    loadEnd=time.time()
```

المنارة للاستشارات
www.manaraa.com

```
print('Load␣time:␣',loadEnd-loadStart)

for j in range(len(modelNames)):
    if num_jobs_creating_map!=1:
        pool=Pool(processes=num_jobs_creating_map)
    fullName=name+modelNames[j]
    model=models[j]

    logging.info("Creating␣Maps␣for␣model␣%s" % name)
    model = GetModel(model, Yscaled, num_jobs_creating_map)

    # If RFE is True, perform it now. (Reduce the features in the model to
        num_features before making predictions.)
    originalFeatures=features.copy()
    if rfe==True:
        rfe, model, features = TryRFE(model, Xscaled, Yscaled, features,
            num_features)
    else:
        model.fit(Xscaled, Yscaled)

    numOutputs = len(layers)

    # Finding which sites to make predictions for:
    inputMap={}
    numSites=np.shape(mapinput)[0]
    argsList1=[]
    results1=0

    poolStart=time.time()
    if num_jobs_creating_map==1:
        results1=[ScaleAndPredict(model,Xscaler,Yscaler,originalFeatures,
            mapinput,numOutputs,mapDim,naturalMap,anthropogenicMap)]
    else:
        for i in range(math.floor(num_jobs_creating_map)):
            numInGroup=int(math.ceil(numSites/(num_jobs_creating_map)))
            if i==num_jobs_creating_map-1:
                inputMap[i]=mapinput[numInGroup*(i):numSites]
            else:
                inputMap[i]=mapinput[numInGroup*(i):numInGroup*(i+1)]
            argsList1.append((model, Xscaler, Yscaler, originalFeatures,
                inputMap[i], numOutputs, mapDim, naturalMap, anthropogenicMap))
        results1=pool.starmap(ScaleAndPredict,(argsList1))
        pool.close()
        pool.join()
```

المنارة للاستشارات
www.manaraa.com

```
poolEnd=time.time()
print('Pooling␣time:␣', poolEnd-poolStart)

if num_jobs_creating_map==1:
    soundmap=results1[0][1]
    if naturalMap==True:
        natMap=results1[0][2]
    if anthropogenicMap==True:
        anthMap=results1[0][3]
else:
    for i in range(num_jobs_creating_map):
        if i==0:
            soundmap=results1[i][1]
            if naturalMap==True:
                natMap=results1[i][2]
            if anthropogenicMap==True:
                anthMap=results1[i][3]
        else:
            soundmap=np.r_[soundmap,results1[i][1]]
            if naturalMap==True:
                natMap=np.r_[natMap,results1[i][2]]
            if anthropogenicMap==True:
                anthMap=np.r_[anthMap,results1[i][3]]
saveStart=time.time()
if (type(Yscaler).__name__ is 'StandardScaler_Multiple' or type(Yscaler).
    __name__ is 'StandardScaler_Single' or type(Yscaler).__name__ is '
    StandardIntensityScaler_Multiple' or type(Yscaler).__name__ is '
    StandardIntensityScaler_Single'):
    soundmap=UseStandardScalersWithNan(soundmap,Yscaler.inverse_transform)
    if naturalMap==True:
        natMap=UseStandardScalersWithNan(natMap,Yscaler.inverse_transform)
    if anthropogenicMap==True:
        anthMap=UseStandardScalersWithNan(anthMap,Yscaler.inverse_transform)
else:
    soundmap=Yscaler.inverse_transform(soundmap)
    if naturalMap==True:
        natMap=Yscaler.inverse_transform(natMap)
    if anthropogenicMap==True:
        anthMap=Yscaler.inverse_transform(anthMap)

if createMap==True:
    SaveMapOutputs(fullName, layers, numOutputs, mapDim, soundmap,
        soundFormats, results_path, rfe, num_features)
if naturalMap==True:
```

www.manaraa.com

```
            newName='Nat_'+fullName
            SaveMapOutputs(newName, layers, numOutputs, mapDim, natMap, soundFormats
                , results_path, rfe, num_features)
        if anthropogenicMap==True:
            newName='Anth_'+fullName
            SaveMapOutputs(newName, layers, numOutputs, mapDim, anthMap,
                soundFormats, results_path, rfe, num_features)
        saveEnd=time.time()
        print('Saving␣time:␣', saveEnd-saveStart)


def ScaleAndPredict(model, Xscaler, Yscaler, features, mapinput, numOutputs, mapDim
    , naturalMap, anthropogenicMap):

    natScaler=NaturalScaler(features)
    natMap=np.array([])
    anthMap=np.array([])
    mapinputNat=np.copy(mapinput)
    if naturalMap==True:
        mapinputNat=natScaler.transform(mapinputNat)

    # Scaling data. If there are nan's, it only matters for the standard scalers.
    # Each feature needs to be scaled the same way as in Xscaler.
        if (type(Xscaler).__name__ is 'StandardScaler_Multiple' or type(Xscaler).
            __name__ is 'StandardScaler_Single' or type(Xscaler).__name__
        is 'StandardIntensityScaler_Multiple' or type(Xscaler).__name__ is '
            StandardIntensityScaler_Single' or type(Xscaler).__name__ is '
            StandardLogScaler_Multiple'):
        mapinput=UseStandardScalersWithNan(mapinput, Xscaler.transform)
        if natMap==True:
            mapinputNat=UseStandardScalersWithNan(mapinputNat, Xscaler.transform)
    else:
        mapinput=Xscaler.transform(mapinput)
        if naturalMap==True:
            mapinputNat=Xscaler.transform(mapinputNat)

    # Creating predictions for the map sites
    soundmap=CreatingMapPredictions(mapinput, numOutputs, model)

    # Creating the predictions for the natural map and anthropogenic map if needed
    if naturalMap==True or anthropogenicMap==True:
        natMap=CreatingMapPredictions(mapinputNat, numOutputs, model)

    if anthropogenicMap == True:
```

المنارة للاستشارات
www.manaraa.com

```
        anthMap = soundmap-natMap

    if naturalMap==False:
        natMap=np.array([])
    if anthropogenicMap==False:
        anthMap=np.array([])
    return [mapDim,soundmap,natMap,anthMap]


def ConvertOutputFormat(soundFormat, fgroups=False):
    dictToReturn={}
    dictToReturn['v']=soundFormat[0]
    dictToReturn['Variable']=soundFormat[1][0]
    dictToReturn['Description']=soundFormat[2][0]
    dictToReturn['Units']=soundFormat[3][0]
    if fgroups==False:
        dictToReturn['hrs']=soundFormat[4][0]
        dictToReturn['freq']=soundFormat[5][0]
        dictToReturn['levels']=soundFormat[6][0]
    else:
        dictToReturn['hrs']=soundFormat[4]
        dictToReturn['freq']=soundFormat[5]
        dictToReturn['levels']=soundFormat[6]

    return dictToReturn

def UseStandardScalersWithNan(inputMat, function):
    if len(np.shape(inputMat))==1:
        inputMat=inputMat.reshape(len(inputMat),1)
    nanMat=np.isnan(inputMat)
    numSites, numCol=np.shape(inputMat)
    inputMatWithoutNan=inputMat[~nanMat.any(axis=1)]
    scaledInputs=0
    if np.shape(inputMatWithoutNan)[0]>0:
        scaledInputs=function(inputMatWithoutNan)
    else:
        print("Only found Nan in this region.")
    outputMat=np.zeros((numSites, numCol))
    outputMat[nanMat.any(axis=1)]=np.nan
    if numCol==1:
        outputMat[~nanMat.any(axis=1),0]=scaledInputs
    else:
        outputMat[~nanMat.any(axis=1)]=scaledInputs
    return outputMat
```

www.manaraa.com

```
def GlobalVariableImportanceMeasures(name, model, features, layers, results_path,
    rfe, num_features):
    # Global variable importance measures:
    # 1. Random forest models:
    try:
        importances=featureImportanceForest(model,features)
        if rfe==False:
            with open('%s_feat_import_RF_%s_%s.txt' % (results_path,name,layers[0]),
                'w') as file:
                file.write(json.dumps(importances, indent=2))
        else:
            with open('%s_feat_import_RF_rfe'+str(num_features)+'_%s_%s.txt' % (
                results_path,name,layers[0]),'w') as file:
                file.write(json.dumps(importances, indent=2))
    except (TypeError,AttributeError):
        # print("Found error in RF importances")
        pass
    # 2. Neural network models:
    try:
        importances=featureImportanceNN(model,features)
        if rfe==False:
            with open('%s_feat_import_NN_%s_%s.txt' % (results_path,name,layers[0]),
                'w') as file:
                file.write(json.dumps(importances, indent=2))
        else:
            with open('%s_feat_import_NN_rfe'+str(num_features)+'_%s_%s.txt' % (
                results_path,name,layers[0]),'w') as file:
                file.write(json.dumps(importances, indent=2))
    except (TypeError,AttributeError):
        # print("Found error in NN importances.")
        pass

def TryRFE(model, Xscaled, Yscaled, features, num_features):
    rfe=True
    try:
        modelRFE=model
        modelRFE=featureSelectionRecursive(model,num_features)
        modelRFE.fit(Xscaled,Yscaled)
        newFeatures=[]
        for i in range(len(modelRFE.support_)):
            if modelRFE.support_[i]==True:
                newFeatures.append(features[i])
        features=newFeatures
```

www.manaraa.com

```
        model=modelRFE
    except (ValueError,RuntimeError,AttributeError):
        print("Cannot␣use␣RFE␣with␣this␣model␣and␣data.")
        rfe=False
        model.fit(Xscaled,Yscaled)
        pass

    return rfe, model, features

def CreatingMapPredictions(mapinput, numOutputs, model):
    nanInput=np.isnan(mapinput)
    numSites=np.shape(mapinput)[0]
    soundmap=np.zeros((numSites, numOutputs))
    for i in range(numSites):

        #if i%1000000==0: # So I know how many sites have been done
            #print(str(i))

        if True in nanInput[i,:]:
            soundmap[i,:]=np.nan
        else:
            soundmap[i,:]=model.predict(mapinput[i,:].reshape(1,-1))

    return soundmap

def SaveMapOutputs(name, layers, numOutputs, mapDim, soundmap, soundFormats,
    results_path, rfe, num_features):
    #print(mapDim)
    #print(soundmap.reshape(mapDim))
    for i in range(numOutputs):
        fgroups=False
        if 'fgroups' in layers[i]:
            fgroups=True
        if 'Anth' in name:
            logging.info("Saving␣anthropogenic␣map␣for␣model␣%s␣with␣layer␣%s" % (
                name, layers[i]))
        elif 'Nat' in name:
            logging.info("Saving␣natural␣map␣for␣model␣%s␣with␣layer␣%s" % (name,
                layers[i]))
        else:
            logging.info("Saving␣map␣for␣model␣%s␣with␣layer␣%s" % (name, layers[i])
                )
        if numOutputs==1:
            soundFormats[0]['v'] = soundmap.reshape(mapDim)
```

www.manaraa.com

```
        else:
            soundFormats[i]['v']=soundmap[:,i].reshape(mapDim)
        if rfe==True:
            sio.savemat(results_path + 'SOUND_' + name + '_rfe' + str(num_features)
                + '_' + soundFormats[i]['Variable'][0] + '.mat', ConvertOutputFormat
                (soundFormats[i], fgroups))
        else:
            sio.savemat(results_path + 'SOUND_' + name + '_' + soundFormats[i]['
                Variable'][0] + '.mat', ConvertOutputFormat(soundFormats[i], fgroups
                ))
```

C.5 PrunedModels.py

```
# Pruned models from each class
# GPR_pruned, KN_pruned, KR_pruned, NN_pruned, GBR_pruned, SV_pruned

from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import *

GPR_pruned = GaussianProcessRegressor(kernel=Matern(), alpha=1e-1, optimizer='
    fmin_l_bfgs_b',
                                 n_restarts_optimizer=10, normalize_y=True,
                                     copy_X_train=True, random_state=1)

GPR_Models = {
    "GPR_pruned": GPR_pruned,
}

from sklearn.neighbors import KNeighborsRegressor

KN_pruned = KNeighborsRegressor(n_neighbors=5,
                       weights='distance',
                       algorithm='auto',
                       leaf_size=30,
                       p=2,
                       metric='minkowski',
                       metric_params=None,
                       n_jobs=1)

KN_Models = {
    "KN_pruned": KN_pruned,
}

from sklearn.kernel_ridge import KernelRidge
```

المنارة للاستشارات
www.manaraa.com

```
KR_pruned = KernelRidge(alpha=0.1,
                        kernel='laplacian',
                        gamma=None,
                        degree=3,
                        coef0=1,
                        kernel_params=None)

KR_Models = {
    "KR_pruned": KR_pruned,
}

from sklearn.neural_network import MLPRegressor

NN_pruned = MLPRegressor(hidden_layer_sizes=(),
                         activation='tanh', # I think this doesn't matter if there are no
                             hidden layers
                         solver='lbfgs', # Works better on small models
                         alpha=50,
                         max_iter=5000,
                         random_state=1, # Should NEVER be None
                         tol=0.000001,
                         verbose=False,
                         warm_start=False,)

NN_Models = {
    "NN_pruned": NN_pruned,
}

from sklearn.ensemble import GradientBoostingRegressor

GBR_pruned = GradientBoostingRegressor(loss='ls',
                                       learning_rate=0.1,
                                       n_estimators=100,
                                       subsample=0.5,
                                       criterion='friedman_mse',
                                       min_samples_split=2,
                                       min_samples_leaf=1,
                                       min_weight_fraction_leaf=0.0,
                                       max_depth=6,
                                       min_impurity_split=1e-07,
                                       init=None,
                                       random_state=0, # Should NEVER be None
                                       max_features='log2',
```

www.manaraa.com

```
                              alpha=0.9,
                              verbose=False,
                              max_leaf_nodes=None,
                              warm_start=False,
                              presort='auto')

RF_Models = {
    "GBR_pruned": GBR_pruned,
}


from sklearn.svm import SVR

SV_pruned = SVR(C=2.0, epsilon=0.1, kernel='rbf',
            degree=3, gamma='auto', coef0=0.0,
            shrinking=False, tol=1e-3, verbose=False, max_iter=-1)

SV_Models = {
    "SV_pruned": SV_pruned,
}
```

C.6 Scalers.py

```
import numpy as np
from sklearn.preprocessing import StandardScaler

# Scalers: StandardScaler_Single, StandardScaler_Multiple, IntensityScaler,
    StandardIntensityScaler_Single,
# StandardIntensityScaler_Multiple, IdentityScaler, LogScaler, NaturalScaler,
    NaturalScalerMaxLimits
class StandardScaler_Single:
        def __init__(self,geo_names):
                self.geo_names=geo_names
                self.scaler = StandardScaler()
                self.called = False

        def transform(self,x):
                if self.called == False:
                        self.scaler=self.scaler.fit(x.flatten().reshape(-1, 1))
                        self.called = True
                x = self.scaler.transform(x.reshape(-1,1)).flatten()
                return x

        def inverse_transform(self,x):
```

www.manaraa.com

```
            if self.called == False:
                self.scaler=self.scaler.fit(x.flatten().reshape(-1, 1))
                self.called = True
            x = self.scaler.inverse_transform(x.reshape(-1,1)).flatten()
            return x

class StandardScaler_Multiple:
        def __init__(self,geo_names):
                self.geo_names=geo_names
                self.scaler = StandardScaler()
                self.called = False

        def transform(self,x):
                if self.called == False:
                        self.scaler=self.scaler.fit(x)
                        self.called = True
                x = self.scaler.transform(x)
                return x

        def inverse_transform(self,x):
                if self.called == False:
                        self.scaler=self.scaler.fit(x)
                        self.called = True
                x = self.scaler.inverse_transform(x)
                return x

# Might not need this.
class IntensityScaler:
        def __init__(self,geo_names):
                self.geo_names=geo_names

        def transform(self,x):
                x=10**(-12)*10**(x/10)
                return x

        def inverse_transform(self,x):
                x[x<=0]=10**(-12)
                x=10*np.log10(x/(10**(-12)))
                return x

# The following two scalers will scale the intensities with the standard scaler,
    and then train on those values.
# At the end, the predicted intensities will be converted back to dB.
class StandardIntensityScaler_Single:
```

المنارة للاستشارات
www.manaraa.com

```
    def __init__(self,geo_names):
        self.geo_names=geo_names
        self.scaler = StandardScaler()
        self.called = False

    def transform(self,x):
        x=10**(-12)*10**(x/10)
        if self.called == False:
            self.scaler=self.scaler.fit(x.flatten().reshape(-1, 1))
            self.called = True
        x = self.scaler.transform(x.reshape(-1,1)).flatten()
        return x

    def inverse_transform(self,x):
        if self.called == False:
            self.scaler=self.scaler.fit(x.flatten().reshape(-1, 1))
            self.called = True
        x = self.scaler.inverse_transform(x.reshape(-1,1)).flatten()
        x[x<=0]=10**(-12)
        x=10*np.log10(x/(10**(-12)))
        return x

class StandardIntensityScaler_Multiple:
    def __init__(self,geo_names):
        self.geo_names=geo_names
        self.scaler = StandardScaler()
        self.called = False

    def transform(self,x):
        x=10**(-12)*10**(x/10)
        if self.called == False:
            self.scaler=self.scaler.fit(x)
            self.called = True
        x = self.scaler.transform(x)
        return x

    def inverse_transform(self,x):
        if self.called == False:
            self.scaler=self.scaler.fit(x)
            self.called = True
        x = self.scaler.inverse_transform(x)
        x[x<=0]=10**(-12)
        x=10*np.log10(x/(10**(-12)))
        return x
```

المنارة للاستشارات

www.manaraa.com

```
class IdentityScaler:
        def __init__(self,geo_names):
                self.geo_names=geo_names

        def transform(self,x):
                return x

        def inverse_transform(self,x):
                return x

class LogScaler:
        def __init__(self, geo_names):
                self.geo_names = geo_names

                # Setting x0 in meters:
                x0_Stream = 3200
                x0_Rail = 13000
                x0_Airport = 24000
                x0_Other = 8000

                stream_names = ['DistStream01','DistStream03','DistStream04']
                rail_names = ['DistRailroads']
                airport_names = ['DistAirpHeli','DistAirpHigh','DistAirpLow','
                    DistAirpMod','DistAirpMoto','DistAirpSea','DistMilitary']
                other_names = ['DistRoadsAll','DistRoadsMaj','DistCoast', '
                    DistWaterbody']

                allNames=[]
                allNames.append(stream_names)
                allNames.append(rail_names)
                allNames.append(airport_names)
                allNames.append(other_names)
                '''
                for name in allNames:
                        if name not in geo_names:
                                print(name+" in LogScaler is incorrectly spelled or
                                    not a feature in this data set.")
                '''
                ind_streams=[]
                ind_rails=[]
                ind_airports=[]
                ind_others=[]
```

www.manaraa.com

```
        for name in stream_names:
            if name in geo_names:
                ind_streams.append(geo_names.index(name))
        for name in rail_names:
            if name in geo_names:
                ind_rails.append(geo_names.index(name))
        for name in airport_names:
            if name in geo_names:
                ind_airports.append(geo_names.index(name))
        for name in other_names:
            if name in geo_names:
                ind_others.append(geo_names.index(name))

        x0 = [False]*len(geo_names)
        for index in ind_streams:
            x0[index] = x0_Stream
        for index in ind_rails:
            x0[index] = x0_Rail
        for index in ind_airports:
            x0[index] = x0_Airport
        for index in ind_others:
            x0[index] = x0_Other

        self.x0 = x0

    def transform(self,x):
        for i in range(len(self.x0)):
            if self.x0[i] != False:
                for j in range(len(x[:,i])):
                    if x[j,i] == 0:
                        x[j,i] = 0.1
                x[:,i] = np.true_divide(x[:,i],self.x0[i])
                x[:,i] = np.log(x[:,i])

        return x

    def inverse_transform(self,x):
        for i in range(len(self.x0)):
            if self.x0[i]==False:
                continue
            else:
                x[:,i]=np.exp(x[:,i])
                x[:,i]=np.multiply(x[:,i],self.x0[i])
```

المنارة للاستشارات
www.manaraa.com

```
            return x


class StandardLogScaler_Multiple:
        def __init__(self, geo_names):
                self.geo_names = geo_names
                self.scaler = StandardScaler()
                self.called = False

                # Setting x0 in meters:
                x0_Stream = 3200
                x0_Rail = 13000
                x0_Airport = 24000
                x0_Other = 8000

                stream_names = ['DistStream01','DistStream03','DistStream04']
                rail_names = ['DistRailroads']
                airport_names = ['DistAirpHeli','DistAirpHigh','DistAirpLow','
                    DistAirpMod','DistAirpMoto','DistAirpSea','DistMilitary']
                other_names = ['DistRoadsAll','DistRoadsMaj','DistCoast', '
                    DistWaterbody']

                allNames=[]
                allNames.append(stream_names)
                allNames.append(rail_names)
                allNames.append(airport_names)
                allNames.append(other_names)

                ind_streams=[]
                ind_rails=[]
                ind_airports=[]
                ind_others=[]

                for name in stream_names:
                        if name in geo_names:
                                ind_streams.append(geo_names.index(name))
                for name in rail_names:
                        if name in geo_names:
                                ind_rails.append(geo_names.index(name))
                for name in airport_names:
                        if name in geo_names:
                                ind_airports.append(geo_names.index(name))
                for name in other_names:
```

المنارة للاستشارات
www.manaraa.com

```
                if name in geo_names:
                        ind_others.append(geo_names.index(name))

        x0 = [False]*len(geo_names)
        for index in ind_streams:
                x0[index] = x0_Stream
        for index in ind_rails:
                x0[index] = x0_Rail
        for index in ind_airports:
                x0[index] = x0_Airport
        for index in ind_others:
                x0[index] = x0_Other

        self.x0 = x0

def transform(self,x):
        for i in range(len(self.x0)):
                if self.x0[i] != False:
                        for j in range(len(x[:,i])):
                                if x[j,i] == 0:
                                        x[j,i] = 0.1
                        x[:,i] = np.true_divide(x[:,i],self.x0[i])
                        x[:,i] = np.log(x[:,i])
        if self.called == False:
                self.scaler=self.scaler.fit(x)
                self.called = True
        x = self.scaler.transform(x)

        return x

def inverse_transform(self,x):
        if self.called == False:
                self.scaler=self.scaler.fit(x)
                self.called = True
        x=self.scaler.inverse_transform(x)
        for i in range(len(self.x0)):
                if self.x0[i]==False:
                        continue
                else:
                        x[:,i]=np.exp(x[:,i])
                        x[:,i]=np.multiply(x[:,i],self.x0[i])

        return x
```

المنارة للاستشارات
www.manaraa.com

```
class NaturalScaler:
        def __init__(self, geo_names):
                self.geo_names = geo_names

                # Setting limits (for no anthropogenic component):
                self.limits={'Built_200m': 0, 'Built_5000m': 0, 'Commercial_200m':
                    0, 'Commercial_5000m': 0, 'Cropland_200m': 0, 'Cropland_5000m':
                    0,
                        'DistAirpHeli': 24000, 'DistAirpHigh': 24000, 'DistAirpLow':
                            24000, 'DistAirpMod': 24000, 'DistAirpMoto': 24000, '
                            DistAirpSea': 24000,
                        'DistMilitary': 24000, 'DistRailroads': 13000, 'DistRoadsAll'
                            : 8000, 'DistRoadsMaj': 8000, 'Extractive_200m': 0, '
                            Extractive_5000m': 0,
                        'ExurbanHigh_200m': 0, 'ExurbanHigh_5000m': 0, '
                            ExurbanLow_200m': 0, 'ExurbanLow_5000m': 0, '
                            FlightFreq_25km': 0, 'Grazing_200m': 0,
                        'Grazing_5000m': 0, 'Industrial_200m': 0, 'Industrial_5000m':
                             0, 'Institutional_200m': 0, 'Institutional_5000m': 0, '
                            MilitarySum_40km': 0,
                        'Mining_200m': 0, 'Mining_5000m': 0, 'Park_200m': 0, '
                            Park_5000m': 0, 'Pasture_200m': 0, 'Pasture_5000m': 0, '
                            PhysicalAccess': 0,
                        'RddAll': 0, 'RddAll_5000m': 0, 'RddMajor': 0, '
                            RddMajor_5000m': 0, 'RecCon_200m': 0, 'RecCon_5000m': 0,
                            'Suburban_200m': 0,
                        'Suburban_5000m': 0, 'Timber_200m': 0, 'Timber_5000m': 0, '
                            Transportation_200m': 0, 'Transportation_5000m': 0, '
                            UrbanHigh_200m': 0,
                        'UrbanHigh_5000m': 0, 'UrbanLow_200m':0, 'UrbanLow_5000m': 0,
                             'VIIRSMaximum_1080m': 0, 'VIIRSMaximum_17280m': 0, '
                            VIIRSMaximum_270m': 0,
                        'VIIRSMaximum_4320m': 0, 'VIIRSMaximum_69120m': 0, '
                            VIIRSMean_1080m': 0, 'VIIRSMean_17280m': 0, '
                            VIIRSMean_270m': 0, 'VIIRSMean_4320m': 0,
                        'VIIRSMean_69120m': 0, 'VIIRSMinimum_1080m': 0, '
                            VIIRSMinimum_17280m': 0, 'VIIRSMinimum_270m': 0, '
                            VIIRSMinimum_4320m': 0,
                        'VIIRSMinimum_69120m': 0, 'WaterHum_200m': 0, 'WaterHum_5000m
                            ': 0, 'WaterNat_200m': 0, 'WaterNat_5000m': 0, 'Wet_200m'
                            : 0, 'Wet_5000m': 0,
                        'PopDensity': 0, 'RoadNoise': 0, 'AviationNoise': 0}
```

المنارة للاستشارات
www.manaraa.com

```
            '''
            for key in self.limits.keys():
                    if key not in geo_names:
                            print(key+" in NaturalScaler is spelled incorrectly or
                                 not a feature in this data set.")
            '''
    def transform(self,x):
            keyNames=list(self.limits.keys())
            for i in range(len(self.geo_names)):
                    if self.geo_names[i] in keyNames:
                            x[:,i]=self.limits[self.geo_names[i]]

            return x

class NaturalScalerMaxLimits:
    def __init__(self, geo_names):
            self.geo_names = geo_names

            # Setting limits (for no anthropogenic component):
            self.limits={'Built_200m': 0, 'Built_5000m': 0, 'Commercial_200m':
                0, 'Commercial_5000m': 0, 'Cropland_200m': 0, 'Cropland_5000m':
                0,
                    'DistAirpHeli': 1.8335*10**5, 'DistAirpHigh': 8.4585*10**5, '
                       DistAirpLow': 2.0567*10**5, 'DistAirpMod': 4.0479*10**5,
                    'DistAirpMoto': 7.0615*10**4, 'DistAirpSea': 7.0615*10**4, '
                       DistMilitary': 2.4496*10**5, 'DistRailroads':
                       1.1998*10**5,
                    'DistRoadsAll': 33725, 'DistRoadsMaj': 65535, '
                       Extractive_200m': 0, 'Extractive_5000m': 0,
                    'ExurbanHigh_200m': 0, 'ExurbanHigh_5000m': 0, '
                       ExurbanLow_200m': 0, 'ExurbanLow_5000m': 0, '
                       FlightFreq_25km': 0, 'Grazing_200m': 0,
                    'Grazing_5000m': 0, 'Industrial_200m': 0, 'Industrial_5000m':
                        0, 'Institutional_200m': 0, 'Institutional_5000m': 0, '
                       MilitarySum_40km': 0,
                    'Mining_200m': 0, 'Mining_5000m': 0, 'Park_200m': 0, '
                       Park_5000m': 0, 'Pasture_200m': 0, 'Pasture_5000m': 0, '
                       PhysicalAccess': 0,
                    'RddAll': 0, 'RddAll_5000m': 0, 'RddMajor': 0, '
                       RddMajor_5000m': 0, 'RecCon_200m': 0, 'RecCon_5000m': 0,
                        'Suburban_200m': 0,
                    'Suburban_5000m': 0, 'Timber_200m': 0, 'Timber_5000m': 0, '
                       Transportation_200m': 0, 'Transportation_5000m': 0, '
                       UrbanHigh_200m': 0,
```

www.manaraa.com

```
                    'UrbanHigh_5000m': 0, 'UrbanLow_200m':0, 'UrbanLow_5000m': 0,
                        'VIIRSMaximum_1080m': 0, 'VIIRSMaximum_17280m': 0, '
                       VIIRSMaximum_270m': 0,
                    'VIIRSMaximum_4320m': 0, 'VIIRSMaximum_69120m': 0, '
                       VIIRSMean_1080m': 0, 'VIIRSMean_17280m': 0, '
                       VIIRSMean_270m': 0, 'VIIRSMean_4320m': 0,
                    'VIIRSMean_69120m': 0, 'VIIRSMinimum_1080m': 0, '
                       VIIRSMinimum_17280m': 0, 'VIIRSMinimum_270m': 0, '
                       VIIRSMinimum_4320m': 0,
                    'VIIRSMinimum_69120m': 0, 'WaterHum_200m': 0, 'WaterHum_5000m
                       ': 0, 'WaterNat_200m': 0, 'WaterNat_5000m': 0, 'Wet_200m'
                       : 0, 'Wet_5000m': 0,
                    'PopDensity': 0, 'RoadNoise': 0, 'AviationNoise': 0}
            '''
            for key in self.limits.keys():
                    if key not in geo_names:
                            print(key+" in NaturalScaler is spelled incorrectly or
                                 not a feature in this data set.")
            '''
        def transform(self,x):
            keyNames=list(self.limits.keys())
            for i in range(len(self.geo_names)):
                    if self.geo_names[i] in keyNames:
                            x[:,i]=self.limits[self.geo_names[i]]

            return x
```

C.7 Validation.py

```
import numpy as np
import time
import logging
from AnalysisTools import featureSelectionRecursive


def RMSE(observed, predicted):
    """
    Calculate root mean square error
    """
    r = observed - predicted
    return np.sqrt(np.mean(r * r))


def MAD(observed, predicted):
```

المنارة للاستشارات
www.manaraa.com

```
    """
    Calculate median absolute deviation
    """
    r = observed - predicted
    return np.median(np.abs(r))


def LOOCV(model, X, Y, rfe, num_features):
    """
    Perform a leave-one-out cross validation
    Returns the predictions of the fit model for each validation fit
    """
    predictions = np.zeros(Y.shape)
    if len(predictions.shape) == 1:
        predictions = predictions.reshape(1, -1).T

    featIndices=[]
    if rfe==True:
        try:
            modelRFE=featureSelectionRecursive(model,num_features)
            modelRFE.fit(X,Y)
            featIndices=[]
            for i in range(len(modelRFE.support_)):
                if modelRFE.support_[i]==True:
                    featIndices.append(i)
            X=X[:,featIndices]
        except (ValueError,RuntimeError,AttributeError):
            print("Cannot use RFE with this model and data.  LOOCV will be 
                calculated without RFE.")
            pass

    m = Y.shape[0]
    indices = np.ones(m, dtype=bool)
    start = time.time()
    for i in range(m):
        indices[i] = False
        model.fit(X[indices], Y[indices])
        predictions[i, :] = model.predict(X[i].reshape(1, -1))
        indices[i] = True
        logging.info("Calculating LOOCV error %i/%i" % (i, m))
    end = time.time()
    logging.info("Time taken = %01f minutes" % ((end - start) / 60))
    return predictions
```

المنارة للاستشارات
www.manaraa.com

C.8 AnalysisTools.py

```
# Analysis Tools

# Feature Importance/Selection Functions (note that some of these only work with
    specific model types):

import numpy as np
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import *
import json
from scipy.stats import pearsonr
from sklearn.feature_selection import RFE
from collections import OrderedDict

def featureImportanceForest(forest_model, features):
        '''
        Find and print the feature importance of all features using the default
            method for
        random forests.
        '''
        importances=forest_model.feature_importances_

        feature_import={}

        for i in range(len(features)):
                feature_import[features[i]]=importances[i]

        feature_import=OrderedDict(sorted(feature_import.items(), key=lambda x:x
            [1],reverse=True))

        return feature_import

def featureImportanceNN(nn_model, features):
        coefs=nn_model.coefs_
        importances={}
        #absCoefs=[abs(x) for x in coefs]
        #print(absCoefs/np.sum(absCoefs))
        #print(features)
        allWeights=np.zeros((len(features),1))
        numLayers=len(coefs)
        numPaths=1
        hiddenLayers=[]
        for i in range(numLayers):
```

www.manaraa.com

```
        numPaths=numPaths*np.shape(coefs[i])[1]
        hiddenLayers.append(np.shape(coefs[i])[1])

for featIndex in range(len(features)):
        allPaths=np.ones((numPaths,1))
        for i in range(numLayers-1):
                index=0
                numToRepeat=1
                for j in range(numLayers):
                        if j>i:
                                numToRepeat=numToRepeat*hiddenLayers[j]
                location=0
                while index<numPaths:
                        if i==0:
                                allPaths[index:index+numToRepeat,0]=allPaths[
                                    index:index+numToRepeat,0]*coefs[i][
                                    featIndex,:].flatten()[location]
                        else:
                                allPaths[index:index+numToRepeat,0]=allPaths[
                                    index:index+numToRepeat,0]*coefs[i].flatten
                                    ()[location]
                        location=location+1
                        if location==len(coefs[i].flatten()):
                                location=0
                        index=index+numToRepeat
        sumPaths=np.zeros((hiddenLayers[0],1))
        numSubPaths=int(numPaths/hiddenLayers[0])
        for i in range(hiddenLayers[0]):
                sumPaths[i]=np.sum(abs(allPaths[i*numSubPaths:(i+1)*
                    numSubPaths]))
        featImport=np.sum(sumPaths[:].flatten()*coefs[0][featIndex,:].
            flatten())
        allWeights[featIndex]=abs(featImport)

if len(features)>1:
        scaledWeights=allWeights/np.sum(allWeights)
        i=0
        for feature in features:
                importances[feature]=scaledWeights[i][0]
                i=i+1
else:
        importances[features[0]]=1

importances=OrderedDict(sorted(importances.items(), key=lambda x:x[1],
```

المنارة للاستشارات
www.manaraa.com

```
            reverse=True))
        #print(importances)
        return importances

def featureSelectionCorrelation(X, features, max_corr=0.9):
        # Runtime warnings created from the 'Mining_200m' feature (all 0's in
            training set)-
        finished=False
        while finished==False:
                restart=False
                for i in range(len(features)):
                        for j in range(len(features)):
                                if abs(pearsonr(X[:,i],X[:,j])[0])>=max_corr and i!=j:
                                        totCorr1=-1
                                        totCorr2=-1
                                        for k in range(len(features)):
                                                totCorr1=totCorr1+abs(pearsonr(X[:,i],X
                                                    [:,k])[0])
                                                totCorr2=totCorr2+abs(pearsonr(X[:,j],X
                                                    [:,k])[0])
                                        if totCorr2>totCorr1:
                                                features.remove(features[j])
                                                X=np.delete(X,j,1)
                                        else:
                                                features.remove(features[i])
                                                X=np.delete(X,i,1)
                                        restart=True
                                        break
                        if restart==True:
                                break
                if restart==False:
                        finished=True

        return X,features

def featureSelectionStd(X, features, max_std):
        numFeat=len(features)
        indToKeep=[]
        for i in range(numFeat):
                if np.std(X[:,i])>max_std:
                        indToKeep.append(i)
        return X[:,indToKeep], [features[i] for i in indToKeep]

def featureSelectionRecursive(model_name, num_features):
```

المنارة للاستشارات

www.manaraa.com

```
    '''
    Recursively selects smaller and smaller sets of features. Only works on
        single
    output models.
    '''
    selector=RFE(model_name, num_features)
    return selector
```

C.9 createFullConus.py

```
import numpy as np
import scipy.io as sio

folderPath='../results/20180315_CONUS_OUTPUT/'
data='SOUND_data1454'
metric='Summer_L90_dBA_day_fgroups'

models=['GPR_pruned','GBR_pruned','KN_pruned','KR_pruned','NN_pruned','SV_pruned']
for model in models:

  nw=sio.loadmat(folderPath+data+'_NW_'+model+'_'+metric+'.mat')
  sw=sio.loadmat(folderPath+data+'_SW_'+model+'_'+metric+'.mat')
  ne=sio.loadmat(folderPath+data+'_NE_'+model+'_'+metric+'.mat')
  se=sio.loadmat(folderPath+data+'_SE_'+model+'_'+metric+'.mat')

  conus=np.zeros((10725,17091))
  conus[0:5362,0:8545]=nw['v']
  conus[0:5362,8545:]=ne['v']
  conus[5362:,0:8545]=sw['v']
  conus[5362:,8545:]=se['v']
  nw['v']=conus

  sio.savemat(folderPath+data+'_'+model+'_'+metric+'.mat',nw)
```

C.10 findMedian.py

```
import numpy as np
import scipy.io as sio

folderPath='../results/20180315_CONUS_OUTPUT/'
data='SOUND_data1447'
metric='Summer_L50_dBA_day'

gbr=sio.loadmat(folderPath+data+'_GBR_pruned_'+metric+'.mat')
```

www.manaraa.com

```
gpr=sio.loadmat(folderPath+data+'_GPR_pruned_'+metric+'.mat')
kr=sio.loadmat(folderPath+data+'_KR_pruned_'+metric+'.mat')
kn=sio.loadmat(folderPath+data+'_KN_pruned_'+metric+'.mat')
nn=sio.loadmat(folderPath+data+'_NN_pruned_'+metric+'.mat')
sv=sio.loadmat(folderPath+data+'_SV_pruned_'+metric+'.mat')

r,c=np.shape(gbr['v'])
allRegion=np.zeros((r,c,6))
allRegion[:,:,0]=gbr['v']
allRegion[:,:,1]=gpr['v']
allRegion[:,:,2]=kr['v']
allRegion[:,:,3]=kn['v']
allRegion[:,:,4]=nn['v']
allRegion[:,:,5]=sv['v']

regMed=np.nanmedian(allRegion,axis=2)
regStd=np.nanstd(allRegion,axis=2)
gbr['v']=regMed
gpr['v']=regStd
sio.savemat(folderPath+data+'_Ensemble_'+metric+'.mat',gbr)
sio.savemat(folderPath+data+'_Std_'+metric+'.mat',gpr)
```

C.11 Sample Data File: Data1008.py

```
# PreProcessing, Load Data
import ParseData
from Scalers import *

layers = ["Summer_L50_dBA_day"]
features_to_remove = []
X, Y, soundFormats = ParseData.LoadData(layers)
X, Y = ParseData.RemoveAnyNans(X, Y)
X, features = ParseData.RemoveListOfFeatures(X, ParseData.geo_names,
    features_to_remove)

Xscaler = StandardScaler_Multiple(features)
Yscaler = StandardScaler_Single(features)
Xscaled = Xscaler.transform(X)
Yscaled = Yscaler.transform(Y)
data_name = "data1008"
data_description = ParseData.data_name + "_Single␣output␣model␣for␣L50␣daytime␣
    levels_all␣features_Standard."
```

المنارة للاستشارات
www.manaraa.com

Bibliography

[1] NPS, "NPS Director's Order ♯ 47: Soundscape Preservation and Noise Management," (2000).

[2] N. A. of Engineering, "Protecting National Park Soundscapes," Washington, DC: The National Academic Press (2013).

[3] D. Weinzimmer, P. Newman, D. Taff, J. Benfield, E. Lynch, and P. Bell, "Human Responses to Simulated Motorized Noise in National Parks," Leis. Sci. **36,** 251–267 (2014).

[4] C. D. Francis *et al.*, "Acoustic Environments Matter: Synergistic Benefits fo Humans and Ecological Communities," Environ. Manage. **203,** 245–254 (2017).

[5] C. I. Merchan, L. Diaz-Balteiro, and M. Soliño, "Noise Pollution in National Parks: Soundscape and Economic Valuation," Landsc. Urban Plan. **123,** 1–9 (2014).

[6] M. E. Beutel *et al.*, "Noise Annoyance is Associated with Depression and Anxiety in the General Population - the Contribution of Aircraft Noise," PLoS ONE 11 (2016).

[7] T. Bodin, M. Albin, J. Ardö, E. Stroh, P. Östergren, and J. Björk, "Road Traffic Noise and Hypertension: Results from a Cross-Sectional Public Health Survey in Southern Sweden," Environmental Health **8,** 38 (2009).

[8] L. Jarup *et al.*, "Hypertension and Exposure to Noise Near Airports: the HYENA Study," Environmental Health Perspectives **116,** 329 (2008).

المنارة للاستشارات

www.manaraa.com

[9] T. Münzel, F. P. Schmidt, S. Steven, J. Herzog, A. Daiber, and M. Sørensen, "Environmental Noise and the Cardiovascular System," J. Am. Coll. Cardiol. **71,** 688–697 (2018).

[10] S. A. Stansfeld *et al.*, "Aircraft and Road Traffic Noise and Children's Cognition and Health: a Cross-National Study," Lancet **365,** 1942–1949 (2005).

[11] S. P. Banbury, W. J. Macken, S. Tremblay, and D. M. Jones, "Auditory Distraction and Short-Term Memory: Phenomena and Practical Implications," Human Factors **43,** 12–29 (2001).

[12] C. D. Francis, C. P. Ortega, and A. Cruz, "Noise Pollution Changes Avian Communities and Species Interactions," Current Biology **19,** 1415–1419 (2009).

[13] H. Slabbekorn and W. Halfwerk, "Behavioural Ecology: Noise Annoys at Community Level," Current Biology **19,** R693–R695 (2009).

[14] B. C. Pijanowski, L. T. Villanueva-Rivera, S. L. Dumyahn, A. Farina, B. L. Krause, B. M. Napoletano, S. H. Gage, and N. Pieretti, "Soundscape Ecology: The Science of Sound in the Landscape," BioScience **61,** 203–216 (2011).

[15] E. P. Derryberry, R. M. Danner, J. E. Danner, G. E. Derryberry, J. N. Phillips, S. E. Lipshutz, K. Gentry, and D. A. Luther, "Patterns of Song across Natural and Anthropogenic Soundscapes Suggest That White-Crowned Sparrows Minimize Acoustic Masking and Maximize Signal Content," PLoS ONE 11 (2016).

[16] G. Buscaino *et al.*, "Temporal Patterns in the Soundscape of the Shallow Waters of a Mediterranean Marine Protected Area," Scientific Reports 6 (2016).

[17] P. A. Hastings and A. Širović, "Soundscapes Offer Unique Opportunities for Studies of Fish Communities," In *Proceedings of the National Academy of Sciences of the United States of America*, **112,** 5866–5867 (2015).

المنارة للاستشارات

www.manaraa.com

[18] L. Ruppé, G. Clément, A. Herrel, L. Ballesta, T. Décamps, L. Kéver, and E. Parmentier, "Environmental Constraints Drive the Partitioning of the Soundscape in Fishes," In *Proceedings of the National Academy of Sciences of the United States of America*, **112,** 6092–6097 (2015).

[19] F. Bertucci, E. Parmentier, G. Lecellier, A. D. Hawkins, and D. Lecchini, "Acoustic Indices Provide Information on the Status of Coral Reefs: An Example from Moorea Island in the South Pacific," Scientific Reports 6 (2016).

[20] S. M. Haver, H. Klinck, S. L. Nieukirk, H. Matsumoto, R. P. Dziak, and J. L. Miksis-Olds, "The Not-So-Silent World: Measuring Arctic, Equitorial, and Antarctic soundscapes in the Atlantic Ocean," Deep Sea Research Part I: Oceanographic Research Papers (2017).

[21] S. Goutte, A. Dubois, and F. Legendre, "The Importance of Ambient Sound Level to Characterise Anuran Habitat," PLoS ONE 8 (2013).

[22] M. Tennesen, "Gauging Biodiversity by Listening to Forest Sounds," Scientific American (2008).

[23] S. J. Pan and Q. Yang, "A Survey on Transfer Learning," IEEE Transactions on Knowledge and Data Engineering **22,** 1345–1359 (2010).

[24] S. Marsland, *Machine Learning: An Algorithmic Perspective*, 2nd ed. (CRC Press, Boca Raton, FL, 2015).

[25] C. M. Bishop, in *Pattern Recognition and Machine Learning*, M. Jordan, J. Kleinberg, and B. Schölkopf, eds., (Springer Science+Business Media, LLC, 2006).

[26] J. R. Quinlan, "Induction of Decision Trees," Machine Learning pp. 81–106 (1986).

[27] L. Breiman, "Random Forests," Machine Learning **45,** 5–32 (October 2001).

المنارة للاستشارات
www.manaraa.com

[28] Cdipaolo96, "File: Gaussian Process Regression.png," retrieved from https://commons.wikimedia.org/w/index.php?curid=47589433 (accessed March, 2016).

[29] R. C. Smith, *Uncertainty Quantification: Theory, Implementation, and Applications* (SIAM Computational Science & Engineering Series, Philadelphia, PA, USA, 2014).

[30] M. C. Kennedy and A. O'Hagan, "Bayesian Calibration of Computer Models," J. R. Statist. Soc. **63,** 425–464 (2001).

[31] D. J. Mennitt, K. Fristrup, K. Sherrill, and L. Nelson, "Mapping Sound Pressure Levels on Continental Scales Using a Geospatial Sound Model," InterNoise13 (2013).

[32] D. Mennitt, K. Sherrill, and K. Fristrup, "A Geospatial Model of Ambient Sound Pressure Levels in the Contiguous United States," J. Acoust. Soc. Am. **135,** 2746–2764 (May 2014).

[33] D. Mennitt and K. Fristrup, "Influential Factors and Spatiotemporal Patterns of Environmental Sound Levels in the Contiguous United States," Noise Control Engr. J. **64,** 342–353 (2016).

[34] D. Xie, Y. Liu, and J. Chen, "Mapping Urban Environmental Noise: A Land Use Regression Method," Environ. Sci. Technol. **45,** 7358–7364 (2011).

[35] P. H. Ryan and G. K. LeMasters, "A Review of Land-use Regression Models for Characterizing Intraurban Air Pollution Exposure," Inhalation Toxicology **19,** 127–133 (2007).

[36] W. J. Galloway, K. M. Eldred, and M. A. Simpson, "Population Distribution of the United States as a Function of Outdoor Noise Level, Volume 2," U.S. Environmental Protection Agency (Washington, D.C.) (1974).

المنارة للاستشارات

www.manaraa.com

[37] Boeing, “Airports with Noise and Emissions Restrictions,” retrieved from https://www.boeing.com/commercial/noise/list.page (accessed February, 2018).

[38] F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python,” JMLR **12,** 2825–2830 (2011).

[39] M. Gevrey, I. Dimopoulos, and S. Lek, “Review and Comparison of Methods to Study the Contribution of Variables in Artificial Neural Network Models,” Ecological Modelling 160 (2003).

المنارة للاستشارات

www.manaraa.com