

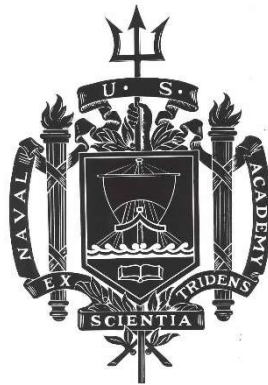
A TRIDENT SCHOLAR PROJECT REPORT

NO. 496

A Machine Learning Model for Prediction of Optical Turbulence in Near-Maritime Environments

by

Midshipman 1/C Christopher D. Jellen, USN



UNITED STATES NAVAL ACADEMY
ANNAPOLIS, MARYLAND

This document has been approved for public
release and sale; its distribution is unlimited.

USNA-1531-2

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) 7/6/20		2. REPORT TYPE		3. DATES COVERED (From - To)	
4. TITLE AND SUBTITLE A Machine Learning Model for Prediction of Optical Turbulence in Near-Maritime Environments				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Jellen, Christopher D.				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Naval Academy Annapolis, MD 21402				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S) Trident Scholar Report no. 496 (2020)	
12. DISTRIBUTION / AVAILABILITY STATEMENT This document has been approved for public release; its distribution is UNLIMITED.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT As a beam propagates, it is subject to fluctuations in the refractive index of air. These effects can be modeled as optical turbulence. Optical turbulence limits the effectiveness of laser-based weapons and communication systems employed by the United States Navy. Models developed to predict optical turbulence through the structure constant C_n^2 are sensitive to absolute air temperature. Existing models have, however, failed to accurately predict the rapid beam attenuation and corresponding high values of C_n^2 observed in maritime and near-maritime environments. In response, data-driven machine learning models were developed to predict the refractive index structure parameter C_n^2 , and to explore the importance of various environmental factors on its prediction. The current study uses 15 months of C_n^2 field measurements collected along an 890 m scintillometer link over the Severn River at the United States Naval Academy. Measures of optical turbulence are complemented by corresponding measurements of 12 environmental parameters. Fully data-driven models were trained, developed, and tested to enhance C_n^2 prediction accuracy in the near-maritime environment. Analysis of these models resulted in better understanding of the relative importance of each environmental parameter in accurately predicting C_n^2 . To our knowledge, this is the first application of purely data-driven machine learning models for predicting C_n^2 in the near-maritime environment.					
15. SUBJECT TERMS optical turbulence, beam propagation, near-maritime, machine learning, data driven model					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES 95	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER (include area code)

U.S.N.A – Trident Scholar project report; no. 496 (2020)

**A MACHINE-LEARNING MODEL FOR PREDICTION OF OPTICAL TURBULENCE
IN NEAR-MARITIME ENVIRONMENTS**

by

Midshipman 1/C Christopher D. Jellen
United States Naval Academy
Annapolis, Maryland

Certification of Advisers Approval

Associate Professor John Burkhardt
Mechanical Engineering Department

Professor Cody J. Brownell
Mechanical Engineering Department

Assistant Professor Charles Nelson
Electrical Engineering Department

Acceptance for the Trident Scholar Committee

Professor Maria J. Schroeder
Associate Director of Midshipman Research

Abstract

As a beam propagates, it is subject to fluctuations in the refractive index of air. These effects can be modeled as optical turbulence. Optical turbulence limits the effectiveness of laser-based weapons and communication systems employed by the United States Navy. Models developed to predict optical turbulence through the structure constant C_n^2 are sensitive to absolute air temperature. Existing models have, however, failed to accurately predict the rapid beam attenuation and corresponding high values of C_n^2 observed in maritime and near-maritime environments. In response, data-driven machine learning models were developed to predict the refractive index structure parameter C_n^2 , and to explore the importance of various environmental factors on its prediction.

The current study uses 15 months of C_n^2 field measurements collected along an 890 m scintillometer link over the Severn River at the United States Naval Academy. Measures of optical turbulence are complemented by corresponding measurements of 12 environmental parameters. Fully data-driven models were trained, developed, and tested to enhance C_n^2 prediction accuracy in the near-maritime environment. Analysis of these models resulted in better understanding of the relative importance of each environmental parameter in accurately predicting C_n^2 . To our knowledge, this is the first application of purely data-driven machine learning models for predicting C_n^2 in the near-maritime environment.

Both parametric and non-parametric models were investigated. A general linear model, regression tree, random forest, and boosted regression tree model were trained on field observations and tuned to improve their predictive power. Compared to linear models, regression trees, random forests, and boosted models demonstrated higher C_n^2 prediction accuracy. The random forest and boosted model were analyzed to determine the relative importance of each measured environmental parameter in predicting C_n^2 . The absolute air temperature, air-water temperature difference, temporal hour weight, and seasonality were significant in making accurate predictions of C_n^2 in the near-maritime environment.

Keywords: optical turbulence, beam propagation, near-maritime, machine learning, data driven model

Acknowledgements

This work could not have been accomplished without the unwavering support and advice of Professor John Burkhardt, Professor Cody Brownell, and Professor Charles Nelson. Further instruction and review from Professors William Traves, Douglas Vanderwerken, Daren Creutz, Daniel Roche, Emily Retzlaff, and CAPT Chris Ruth, USN, was instrumental throughout the research process.

The authors would also like to thank the United States Naval Academy Mechanical Engineering, Electrical Engineering, Oceanography and Mathematics Departments for their assistance in data collection, model critique, and editorial review. Finally, we would like to thank the Trident Committee and Professor Maria Schroeder for their generous support.

This research was funded by the Office of Naval Research, The Joint Technology Office, and the United States Naval Academy Trident Program.

Table of Contents

1	Introduction	5
2	Background.....	6
2.1	Data-driven Models for <i>Cn2</i>	10
3	Experimental Setup.....	11
3.1	Data Sources.....	12
3.2	Data set preparation.....	17
3.3	Data Set Overview	19
4	Modeling.....	23
4.1	General Linear Model	24
4.1.1	Training	25
4.1.2	Results and Discussion	26
4.2	Regression Trees	28
4.2.1	Training	28
4.2.2	Results and Discussion	29
4.3	Random Forest	33
4.3.1	Training	34
4.3.2	Results and Discussion	35
4.4	Boosted Model	42
4.4.1	Training	42
4.4.2	Results	43
5	Model Comparison	49
5.1	Residuals	50
5.2	Model Interpretability	54
6	Relative Feature Importance.....	55
6.1	Generating Feature Importance Metrics.....	55
6.2	Results	56
6.3	Discussion	57
7	Conclusion	58
8	References	60
	Appendices.....	63

Appendix A: Hourly Averaged Data Set MATLAB Script	63
Appendix B: 10-minute Averaged Data Set MATLAB Script	69
Appendix C: Hourly Averaged General Linear Model Training	76
Appendix D: 10-minute Averaged General Linear Model Training	78
Appendix E: Hourly Averaged Random Forest Model Training	80
Appendix F: 10-minute Averaged Random Forest Model Training	84
Appendix G: Hourly Averaged Boosted Model Training	88
Appendix H: 10-minute Averaged Boosted Model Training	92

1 Introduction

Atmospheric optical turbulence has a deleterious effect on the operational capabilities of naval laser weapons, power transmission, and communication systems. Optical turbulence may result in significant beam spread beyond the diffractive limit, and in optical scintillation at the target [1, 2, 3, 4, 5, 6]. For naval weapon and power transmission systems, this increases the time-on-target required to achieve desired effects, and for communication systems this can result in lost data transmission [5]. The impact of atmospheric factors on laser beam propagation motivates the need for a deeper understanding of the relationships between environmental parameters and optical turbulence.

Optical turbulence is the result of refractory fluctuations along the path of an electromagnetic beam, caused by the turbulent mixing of temperature variations in the atmosphere [5, 6, 7]. As the beam passes through regions of varying temperature, and therefore varying refractive index, its wave front becomes perturbed. This causes additional beam spread, increased variance, and a reduction in irradiance on target [5, 6, 7]. Because the turbulent structure of the atmosphere depends on both large-scale forcing and on local conditions, laser propagation will also depend indirectly on a wide range of parameters that influence the spatial structure of the propagation medium [5, 6, 7].

To form semi-empirical models for laser propagation based on environmental variables, theories of optical propagation through random media are combined with physics-based models of atmospheric turbulence and heat transfer [5, 6, 7]. These models are highly effective in high-altitude environments where optical turbulence is lower and local conditions are less significant, but predictions become inaccurate close to the ground [1, 2, 3, 8]. Similar physics-based models have been employed to predict optical turbulence in low-altitude terrestrial environments, which may rely on strong diurnal variations [1, 2]. The relationships between environmental parameters and optical turbulence is less well-defined in low-altitude maritime environments. A wide variety of environmental parameters which are often neglected by overland as well as high altitude turbulence models can have a material impact on optical turbulence in the low-altitude maritime environment [3, 8, 10]. One Naval Postgraduate School study suggested that other environmental factors such as the properties of the ocean below the propagation path made current models inaccurate for maritime environments [8]. Further studies by the Naval Research Laboratory's Chesapeake Bay Detachment suggest that the temperature difference between air and water, as well as the wind speed along the propagation path, may affect optical turbulence [10].

An improved understanding of which environmental parameters are most important for optical turbulence prediction could enhance the accuracy of current models in the low-altitude maritime environment. This understanding could be further leveraged to create a framework for generating data-driven predictions for optical turbulence in a wide variety of microclimates. Rather than using a universal model, a machine learning based model framework could be trained on local data to generate improved predictions for C_n^2 . To our knowledge, the data-driven modeling techniques developed in machine learning have yet to be applied in exploring the physical interactions which generate optical turbulence in the near-maritime environment.

2 Background

In maritime environments, laser beam propagation is affected by many atmospheric factors. As a laser beam propagates along a path, it is subject to beam spreading, scattering, and absorption [1, 2, 3, 4, 5, 6]. Beam spreading and scattering result in rapid beam attenuation in maritime and near-maritime environments. As the beam wave front experiences distortion, its intensity, phase, and polarization, are affected by the local atmosphere [1, 3, 5, 6]. Near-maritime environments constitute a complex atmospheric medium. In this complex medium, the random fluctuations in the index of refraction lead to beam scintillation [4,5,6]. These factors result in an overall reduced effectiveness of laser irradiance on target. Adding to the complexity of predicting laser irradiance are rapid temporal and spatial changes in atmospheric factors. The atmospheric parameters affecting the irradiance of the incident beam change along the propagation path in both space and time [1, 2, 3, 5, 6].

One key atmospheric parameter for laser propagation is the *refractive index* of air. The refractive index describes the speed at which electromagnetic radiation propagates through a medium [5, 6]. The more optically-dense a medium, the more it affects local wave propagation [5, 6]. Specifically, as the beam wave front becomes distorted through atmospheric turbulence, constructive and destructive interference occurs. The primary driver of this distortion is from random *fluctuations*, or rapid changes, in the index of refraction. At optical frequency, these random fluctuations are primarily driven by the absolute temperature difference between different regions of the atmosphere. Contributions due to pressure and humidity of the atmosphere along the beam path are also noted in prior studies [1, 2, 3]. These effects can be extreme at low altitudes in a maritime or near-maritime environment where the refractive index of air can change rapidly over small distances and time intervals [3].

In order to predict the effect of atmospheric turbulence on beam propagation, the local atmosphere can be considered locally homogenous [5]. With this assumption, changes in the refractive index of the atmosphere can be expressed using equation (1) [5, 6].

$$n(\hat{r}, t) = n_0 + n_1(\hat{r}, t) \quad (1)$$

In equation (1), the local refractive index of the atmosphere n is defined to be the mean index of refraction for non-turbulent atmosphere n_0 with the added turbulent effects given by n_1 , which is a function of both the propagation path $\hat{r}$ and time t . This is the Reynolds decomposition for turbulent flows applied to the refractive index. Normalizing the mean index of refraction n_0 to 1 and applying assumptions of temporal homogeneity discussed in [5, 6], the time dependence in the local refractive index is removed. The resulting equation (2) approximates the local refractive index of the atmosphere at spatial coordinate $\hat{r}$ as a function of the propagation path [5, 6].

$$n(\hat{r}) = 1 + n_1(\hat{r}) \quad (2)$$

In equation (2), $n(\hat{r})$ has been normalized by its mean value. The behavior of the refractive index is tied to the physical properties and interactions of the local atmosphere. Prior studies suggest that these fluctuations in the local index of refraction are related to corresponding temperature and pressure fluctuations [6]. For optical and infrared wavelengths, this relationship is expressed in equation (3) [6].

$$n(\hat{r}) \cong 1 + 79 \times 10^{-6} \frac{P(\hat{r})}{T(\hat{r})} \quad (3)$$

In equation (3), the structure of the local index of refraction n is predicted from the pressure P and the temperature T along the path $\hat{r}$. Prior studies suggest that local fluctuations in pressure are generally negligible [6]. As a result, the random fluctuations in the index of refraction occur primarily due to temperature effects. Different regions in the local atmosphere along a propagation path vary in temperature [1, 2, 3, 5, 6]. These regions of varying temperature are described as *eddies*. The temperature gradient between eddies forces turbulent mixing, resulting in fluctuations of the local temperature. A diagram illustrating eddies in the atmosphere of a beam propagation path is shown in Figure 1.

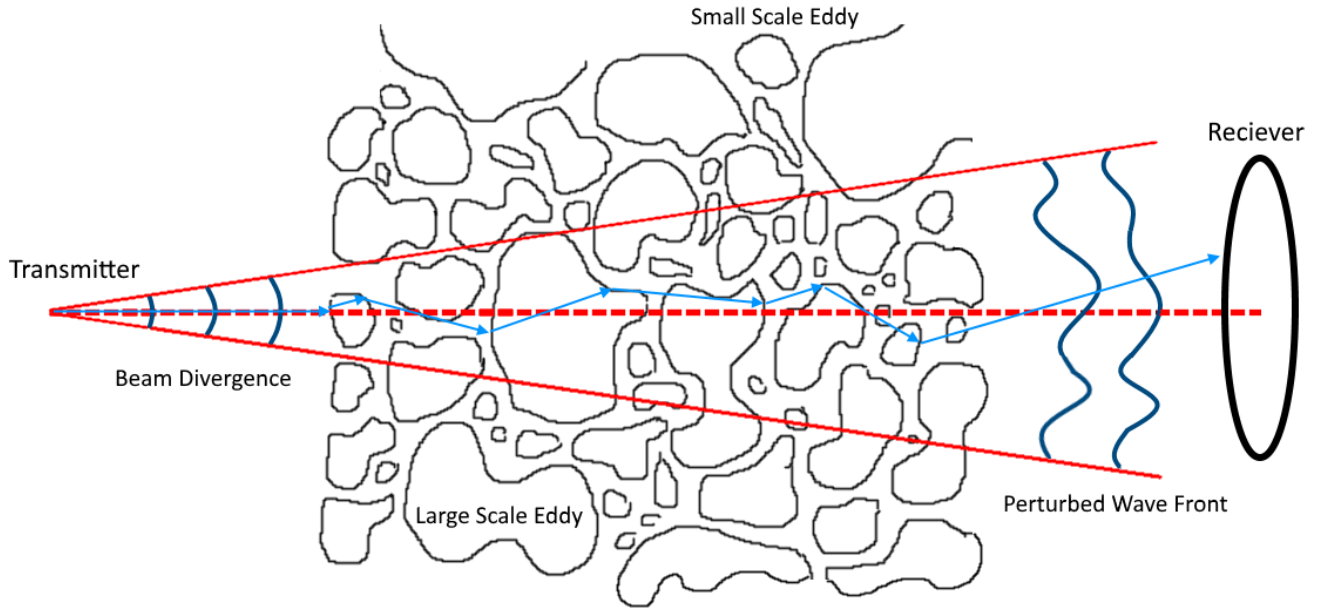


Figure 1: Scintillometer diagram used to measure losses due to turbulent atmospheric characteristics. [7]

In Figure 1, closed curves are taken to represent eddies of various size within the medium between the transmitter and receiver. The small eddies of size l_o and large eddies of size L_o modify beam propagation differently through differences in their refractive index [3, 5, 6, 8]. Both contribute to fluctuations in beam irradiance along the path length L . The number and size of eddies changes with time, which causes variance in the measured beam irradiance at the target receiver [7]. The large eddies are generated by a combination of wind shear and convection. These eddies are unstable and split into smaller eddies as they travel through the atmosphere.

Turbulent mixing between these eddies results in fluctuations of the local refractive index, which can be quantified using a structure function [6]. The structure function for the refractive index

describes the magnitude of differences in the local refractive index between neighboring points, with a separation distance bounded by the outer eddy scale L_0 and an inner scale l_0 . This structure function is given in equation (4) [5, 6].

$$D_T(R) = \langle (T_1 - T_2)^2 \rangle \quad (4)$$

The structure function in equation (4) is useful for describing local atmospheric turbulence as a random field [6]. Two key assumptions are typically made in the application of the structure function. The first is *statistical homogeneity*, which assumes that correlations between random fluctuations of the local refractive index between two points are independent of the locations chosen, and that the mean value of the field is constant [6]. The second key assumption is that of *statistical isotropy*, which states that correlations between the random fluctuations of the refractive index between two points depend only on the magnitude of separation between the points. With these assumptions, dimensional arguments reveal that the structure function for local temperature fluctuations between outer scale L_0 and inner scale l_0 must scale with separation distance R and on a constant C_T^2 that characterizes the magnitude of the differences. This relationship is given in equation (5) [1, 2, 3, 5, 6, 8].

$$D_T(R) = \langle (T_1 - T_2)^2 \rangle = \begin{cases} C_T^2 l_0^{-\frac{4}{3}} R^2 & 0 \leq R \ll l_0 \\ C_T^2 R^{2/3} & l_0 \leq R \ll L_0 \end{cases} \quad (5)$$

In equation (4), C_T^2 is the temperature structure constant, and T_1 and T_2 denote the temperature at the two points separated by R . In the range above the large scale eddies of length L_0 , the field of refractive index fluctuations inherits the properties of homogeneity and isotropy, allowing the field to reduce to a function of only scalar distance R [4,5]. The structure function describing statistically homogenous isotropic turbulence is given by equation (6) [6].

$$D_n(R) = \langle (n_1 - n_2)^2 \rangle = \begin{cases} C_n^2 l_0^{-\frac{4}{3}} R^2 & 0 \leq R \ll l_0 \\ C_n^2 R^{2/3} & l_0 \leq R \ll L_0 \end{cases} \quad (6)$$

In equation (6), C_n^2 is the index of refraction structure constant, with units of $[m^{-\frac{2}{3}}]$. Using the relationship in equation (3), and the assumption of statistical homogeneity and isotropy, the constant C_n^2 can be calculated as in equation (7) [2, 3, 5, 6, 8].

$$C_n^2 = \left(79 \times 10^{-6} \left[\frac{K}{mbar} \right] \frac{P}{T^2} \right)^2 C_T^2 \quad (7)$$

In equation (7), temperature T is measured in $[K]$, pressure P is in $[mbar]$, and C_T^2 has units of $[K^2 m^{-\frac{2}{3}}]$. The index of refraction structure constant C_n^2 is a measure of the intensity of the fluctuations in the refractive index of the atmosphere. The temperature structure function in equations (4) and (5) can be approximated using the mean square temperature difference between measurements taken by two thermometers along the beam propagation path. These assumptions of path invariance allow for approximation of the temperature structure parameter using point measurements of environmental parameters. While these assumptions simplify the

process of predicting C_n^2 , they could contribute to the overestimation of the intensity of optical turbulence in low-altitude maritime environments [8]. In these environments, a wide range of environmental parameters may be required to accurately predict the intensity of optical turbulence.

To verify the accuracy of these assumptions, the refractive index structure parameter can be determined experimentally. Path measures of the variance in beam irradiance are normalized to generate a scintillation index σ_I^2 , defined in equation (8) [1, 2, 5, 6, 8].

$$\sigma_I^2 = \frac{\langle I^2 \rangle - \langle I \rangle^2}{\langle I \rangle^2} \quad (8)$$

The scintillation index is thus a path-dependent measurement of the normalized variance in irradiance along a propagation path. In equation (8), I is the irradiance of the beam at the target at a given time. The brackets in equation (8) denote time averaging, which assumes ergodicity [2, 3, 5, 6]. The scintillation index is used to calculate C_n^2 using the relationship in equation (9) [5,6].

$$\sigma_I^2 = 1.23 C_n^2 k^{\frac{7}{6}} L^{\frac{11}{6}} \quad (9)$$

In equation (9), the path length L and wave number k of a beam are related to the observed scintillation index through the structure constant C_n^2 . Experimentally, a scintillometer is used to measure the variance in irradiance of a beam at a target.

Computing C_n^2 directly can be achieved through computation of the temperature structure function. This process relies on knowledge of local temperature within each of the eddies along the path. Approximations for the temperature structure function can be made through temperature measurement of large and small scale eddies along the propagation path. This process is rarely done experimentally, as small scale eddies are typically only 1-2 mm across in a near-maritime environment [5, 6]. Rather than measuring the absolute temperature difference between different eddies directly, similarity theories employ the use of mean flow characteristics along the propagation path. Monin–Obukhov similarity (MOS) theory is used to describe flow behavior for use in C_n^2 prediction [2, 5, 9]. MOS theory calculates average properties along a path by assuming parameters such as temperature, humidity, and wind speed are path-invariant. Using MOS theory, C_n^2 is predicted using a combination of temperature and humidity effects, where temperature and humidity are modeled with structure parameters from bulk measurements assumed to be invariant along the beam propagation path [2, 5, 9].

These assumptions of path invariance allow for approximation of the temperature and humidity structure parameters using point measurements of environmental parameters. While these assumptions simplify the process of predicting C_n^2 , they could contribute to the overestimation of the intensity of optical turbulence in low-altitude maritime environments [8, 10, 11, 12]. In these environments, a wider range of environmental parameters than those used in MOS theory may be required to accurately predict the intensity of optical turbulence.

2.1 Data-driven Models for C_n^2

In order to better capture the physical relationships which govern beam propagation in the near-maritime environment, data driven machine learning models were trained, developed, tested, and investigated using a wide range of measured local environmental parameters. Machine learning models offer two potential advantages over current bulk atmospheric models. First, they do not force assumptions about physical relationships; they instead allow measured data to generate predictions [13, 14]. Additionally, machine learning models can be adapted to better capture interactions between parameters.

Data driven models are a class of machine learning models based on analyzing data collected about a system in order to develop insight into the relationships within the system, without the use of explicit physical relationships known before model development [13, 15]. Effective data-driven modeling relies on robust data sets. The relationships between parameters within these sets can be determined through model training. In order to develop data driven models, a data set containing observations of *predictor variables*, in this case local atmospheric, oceanographic, and temporal information is collected. Additionally, the corresponding observation of the *response variable* is collected, which the model seeks to predict. The refractive index structure function constant C_n^2 is taken as the response variable in this study. Typical values of C_n^2 range from 1×10^{-16} to 1×10^{-12} in the near-maritime environment, but can get as low as 1×10^{-19} at higher altitudes. Due to the wide range of expected C_n^2 values, the base 10 logarithm of C_n^2 observations was used when training and evaluating models.

Machine learning models can be categorized as parametric, or non-parametric. Parametric models produce a closed form estimate for a response variable, with all model effects captured in a fixed number of parameters. These models often seek the optimal coefficients or weights for each parameter [13]. A simple example of a parametric model is linear regression in which the regression coefficients are the model parameters. These models have a fixed number of terms corresponding to the number of environmental parameters collected. In contrast, non-parametric models are a class of models which do not seek to fit data to a model with a fixed number of terms. Model size in non-parametric models is adaptive, allowing models to grow or shrink in complexity to achieve reasonable fits on training data [13, 14, 16, 17]. Regression trees are a key non-parametric model applied for local C_n^2 prediction, and acts as the foundation for boosted regression tree models and random forests.

3 Experimental Setup

In order to investigate optical turbulence in the low-altitude maritime environment, an 890 m propagation path was established over the Severn River at the United States Naval Academy in Annapolis, Maryland. A scintillometer was used to establish a ground truth measure of optical turbulence, C_n^2 , with which to compare model predictions. The scintillometer link was approximately 2 m over the surface of the water. More than 95% of the length of the propagation path was over water. This propagation environment is characterized as “near-maritime” as the local atmosphere is affected by the land mass on either side of the path [11, 12, 18]. As the link was not over open ocean, tidal changes were modest and the path’s elevation over water was within 1 m of its starting elevation for the duration of the data collection period. An overview of the scintillometer propagation path, as well as the location of various measurement instruments, is shown in Figure 2.

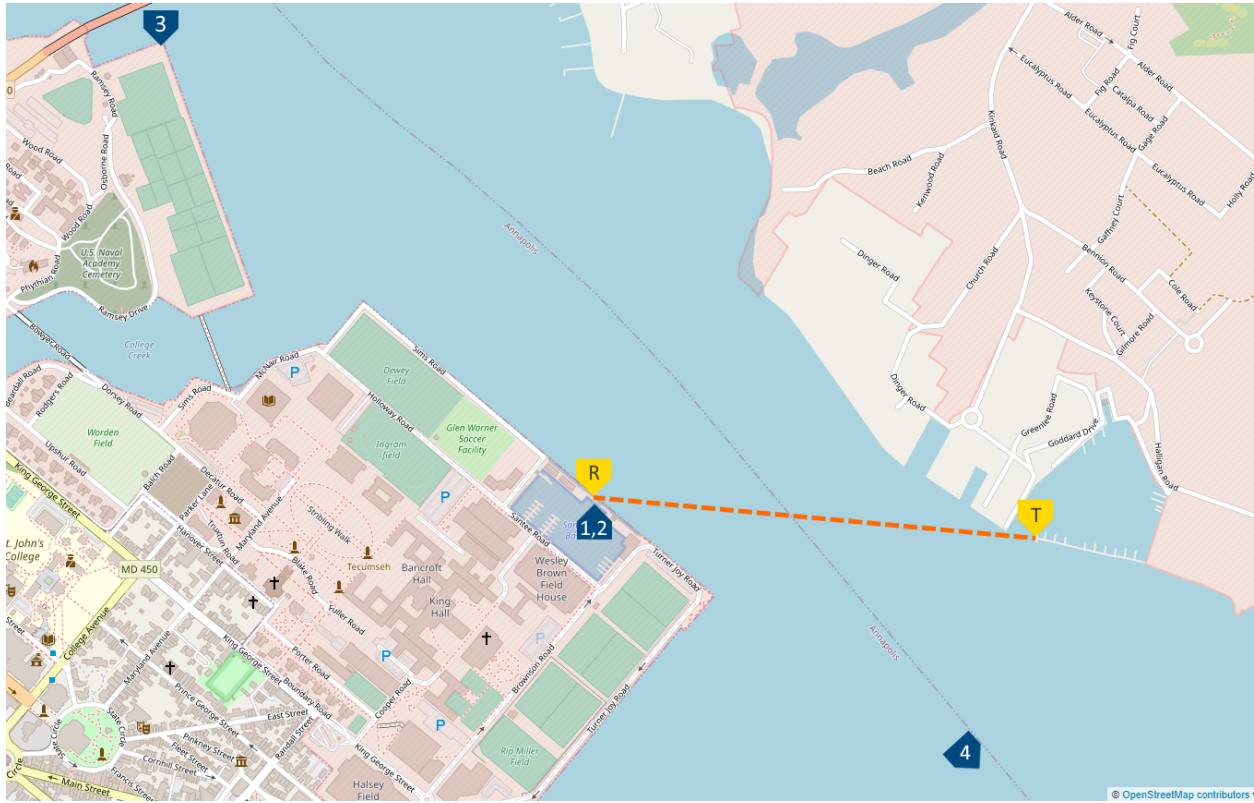


Figure 2: 890 m Scintillometer Link Propagation Path [19].

The link in Figure 2, beginning with a transmitter labeled T and ending with the receiver labeled R , provided a measure of optical turbulence for the duration of the study. In addition to collecting measured readings of optical turbulence through C_n^2 , a wide variety of local environmental parameters were collected. In Figure 2, location 1 and 2 indicate the United States Naval Academy Watershed Observatory and a local Davis weather station placed for the purpose of this study. Location 3 is a National Oceanographic and Atmospheric Association weather station, and location 4 points in the direction of a National Data Buoy Center (NDBC) data buoy. Accurate local environmental parameter measures were essential to inform model predictions.

The focus on local environmental parameters to better capture the physical interactions which affect optical turbulence along a path was informed by prior literature. These studies suggested that, in addition to the air temperature, pressure, and humidity which are incorporated in some current models, the air-water temperature difference and wind speed could be useful in predicting C_n^2 . A table of the nine parameters collected is given in Table 1.

Table 1: Local Environmental Parameters with Their Units and Descriptions.

Parameter	Description	Units	Source
C_n^2	The refractive index structure constant C_n^2 describes the extent of optical turbulent effects on beam propagation. It is path dependent, and reported as a path average.	$m^{-\frac{2}{3}}$	Scintillometer
Air Temperature	Air temperature described the kinetic energy of the gases which make up the local atmosphere.	[C]	Thermometer
Relative Humidity	The relative humidity is the ratio of water vapor in the local atmosphere to the maximum possible water vapor content. This parameter is dependent on air temperature.	[%]	Hygrometer
Wind Speed	The relative speed of local wind to a fixed point on the propagation path.	[m/s]	Anemometer
Air Pressure	The force of the air column above a local propagation path.	[hPa]	Barometer
Rain	The height of rain fallen over a 10 minute interval.	[cm]	Rain Gauge
Solar Radiation	The solar irradiance above the local propagation path.	$[\frac{W}{m^2}]$	Pyranometer
UV Index	A categorical parameter describing the intensity of short-wavelength radiation from the sun.	-	Pyranometer
Air Density	The density of the local atmosphere is related to its pressure and the presence of local aerosols. Aerosols include small solid or liquid particles suspended in the air such as dust or water droplets.	$[\frac{g}{m^3}]$	Barometer
Air-Water Temperature Difference	The difference in temperature between the air and the water below it can describe the extent of turbulent mixing driven by temperature structure function instability.	[C]	Thermometer(s)

The environmental parameters in Table 1 include both atmospheric and oceanographic features, measured in the local environment near the scintillometer link from a variety of data sources. The data sources used had different sampling rates and proximity to the scintillometer link. Both 10-minute-averaged measurements and hourly measurements were collected. The higher-frequency data was collected using local weather stations, predominantly from the Severn River Watershed Observatory at the United States Naval Academy. Hourly averaged data was obtained from nearby weather stations and ocean data buoys, 2 km and 10 km from the link respectively. Further detail on the data sources is provided in section 3.1.

3.1 Data Sources

The first step in generating a data set to train models which estimate optical turbulence along the 890 m link was to establish an experimental measurement of C_n^2 . Field measurements of C_n^2 are taken using a scintillometer, a device which measures the modulation of light along the optical path [20]. A scintillometer senses turbulence between an optical transmitter and receiver. A Scintech BLS450 Large Aperture Scintillometer was used to measure C_n^2 along the path in Figure 3 from January 2019 until March 2020. The transmitting unit of the scintillometer was

established at the edge of the Yard Patrol Craft Basin on the eastern shore of the Severn River shown in Figure 3. The receiving unit was placed on the wall of the Santee Basin, at the western edge of the Severn River. Built-in LED arrays in the transmitting and receiving units were used to align the scintillometer link. The transmitting and receiving units of the scintillometer link are displayed in Figure 3.

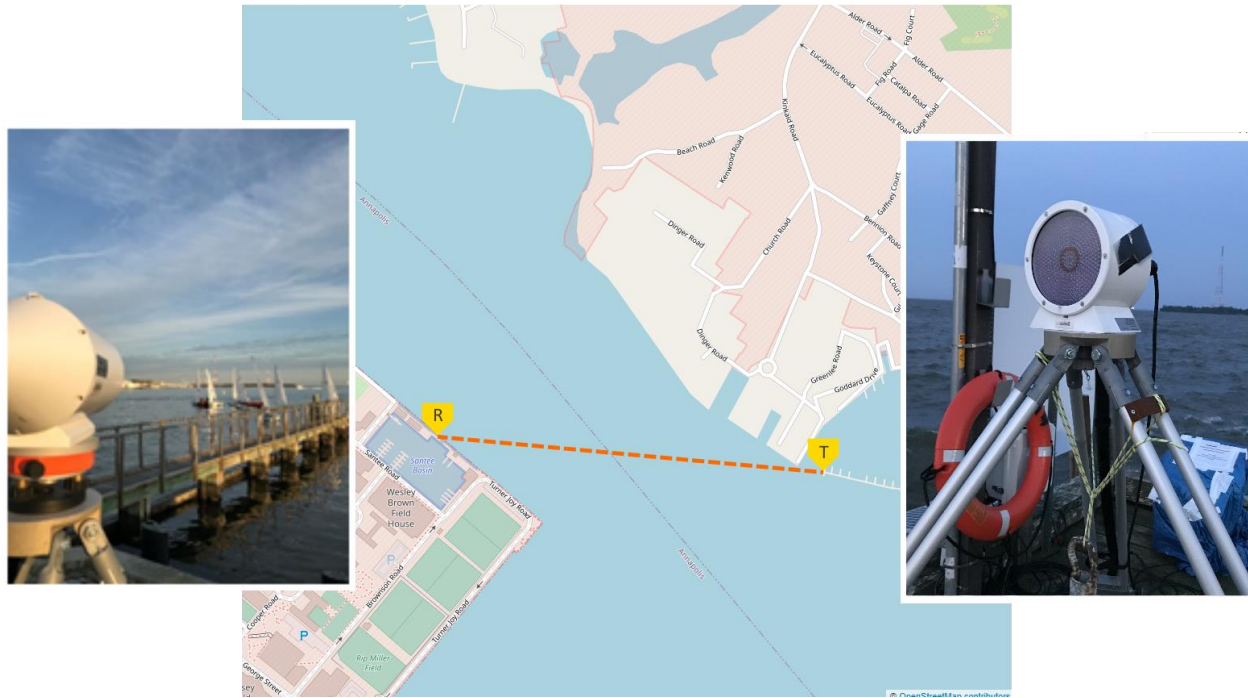


Figure 3: Transmitting and Receiving Units for the 890 m Scintillometer Link [19].

The scintillometer link was established such that the receiver was in the same location as the United States Naval Academy's Severn River Watershed Observatory [21]. This data source is maintained by the United States Naval Academy Oceanography Department, and includes a variety of instruments to measure both meteorological and oceanographic parameters. A Vaisala-WXT536 weather station is fixed approximately 10 m above the Severn River, which reports a variety of atmospheric measurements at 2 or 4-minute intervals [21]. The wind speed, air temperature, relative humidity, atmospheric pressure, and rain measurements from this station formed the core of the environmental parameter data set [21, 22]. The local water temperature was also measured using a Sontek SL500 side-look ADCP current profiler located approximately 2 m below the water surface. This instrument recorded the water temperature every 15 minutes [21]. Both the weather station and current profiler recorded measurements from May 2nd 2019 until November 19th 2019. A diagram of the local weather station setup, including measurement heights, is provided in Figure 4.

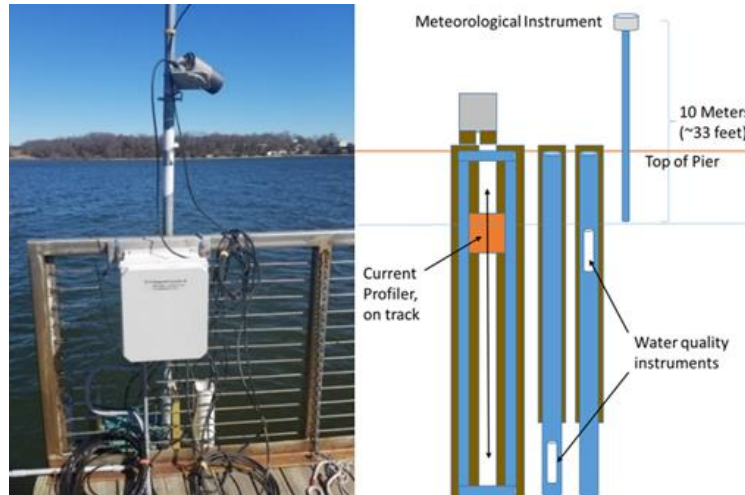
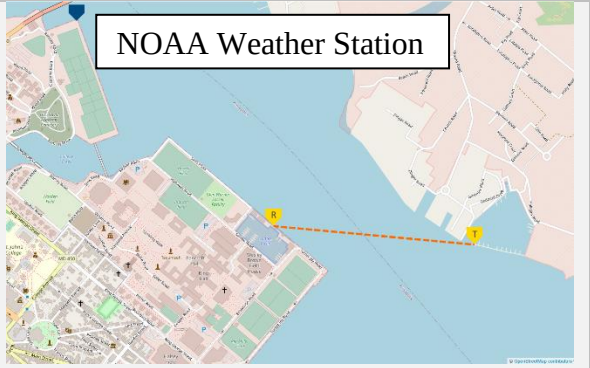


Figure 4: USNA Severn River Watershed Observatory Diagram [21].

To supplement the meteorological measurements taken at the Severn River Watershed Observatory, data from the NOAA WBAN 13752 weather station was used. This weather station is located approximately 2 km from the scintillometer link [18]. This station recorded wind speed, air temperature, relative humidity, atmospheric pressure, and rain measurements to complement those measurements taken at the Watershed Observatory weather station [23]. Additionally, the NOAA weather station measured the local visibility in nautical miles. This data was available from January 4th 2019 until March 31st 2020 in quality-controlled hourly intervals [23]. The NOAA quality controls involved averaging measurements taken over the course of each hour to report an overall measurement for a given time stamp [23]. The reported atmospheric parameters with units of measure, time interval, and location for the NOAA weather station are given in Table 2.

Table 2: NOAA Weather Station WBAN 13752, with Units, Time Interval, and, Location [19].

Parameter	Measurement Interval	Units	Source:
Dew Point Temperature	Observations reported hourly using intermittent averaging.	[°F, °C]	NOAA Weather Station WBAN 13752 Date Range: January 1 st 2019 to 31 st March 2020 Latitude: 38.9913 °N. Longitude: -76.4891 °W. Elevation from Ground: 6 ft.
Atmospheric Pressure	Observations reported hourly using intermittent averaging.	[bar]	
Hourly Pressure Change	Observations reported hourly based on the observed change in reported atmospheric pressure.	[bar]	
Hourly Rainfall	Observations reported hourly, total rainfall for the duration reported.	$[\frac{cm}{hr}]$	
Relative Humidity	Observations reported hourly intermittent averaging.	[%]	

Visibility	Observations reported hourly.	[nm]	
Wind Direction	Observations reported hourly, none listed when local relative wind has no clear and consistent direction.	$[\frac{m}{s}]$	
Average Wind Speed	Observations reported hourly using extreme outlier rejection and intermittent averaging.	$[\frac{m}{s}]$	
Gust Wind Speed	Observations reported hourly, highest wind speed with multiple observations reported.	$[\frac{m}{s}]$	

In addition to the water temperature data collected locally, measurements taken from a National Data Buoy Center (NDBC) station were used. The NDBC station at Thomas Point Light in the vicinity of the scintillometer link measured the water temperature approximately 1 m below the water's surface [24]. This station was fixed in place approximately 10 km from the scintillometer link [18]. Similar to the NOAA weather station, this observation station reported quality controlled hourly readings [24]. Multiple measurements were taken in each time interval to produce an hourly observed local water temperature. This hourly data was available from January 4th 2019 until March 31st 2020 [24].

Two additional temporal parameters were added to the data set before final standardization and analysis. These temporal parameters were collected by the internal clock of the receiver of the scintillometer link. In an attempt to capture the potential relationship between optical turbulence and sunlight, the temporal hour weight was calculated for each observation of C_n^2 . The temporal hour weight is a measure of the expected effect of solar radiation on local optical turbulence. It is obtained by dividing the difference between the time of each observation and local sunrise time by 1/12 the time between local sunrise and sunset, as in equation (10) [5, 6].

$$W_{th} = \frac{12(t_{observed} - t_{sunrise})}{t_{sunset} - t_{sunrise}} \quad (10)$$

The local sunrise and sunset were recorded by the USNA Oceanography Department sunrise data set [2,6, 25]. Additionally, seasonality was measured as the fraction of the year elapsed at the time of each C_n^2 scintillometer observation using the internal scintillometer clock. This measure of seasonality was used as a continuous proxy for the time of year in which the observation was taken. For the time in decimal days of an observation since January 1st, denoted d_{oy} , the seasonality was calculated using equation (11).

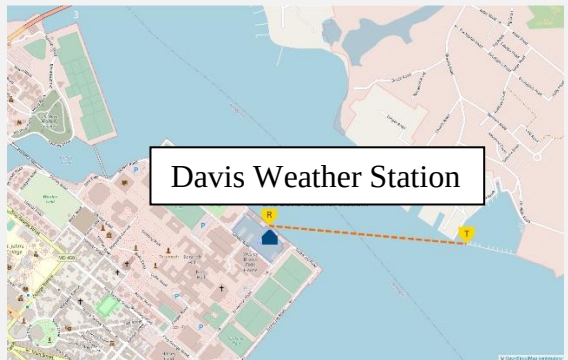
$$S = \frac{d_{oy}}{365.25} \quad (11)$$

A temporal seasonality term was included in an attempt to capture potential seasonal dependence of C_n^2 in the near-maritime environment, which has been suggested by some literature [26].

The data collected from the scintillometer link, Severn River Watershed Observatory, NOAA weather station, NDBC buoy, and scintillometer clock were used to create an hourly-averaged data set for model training. These data sources measured their respective environmental parameters from December 2019 through March 2020. As this data set spanned 15 months of observation, it was used as the primary data set for model development and feature importance analysis.

A secondary data set was developed which used only environmental parameter measurements collected at the receiver of the scintillometer link. Rather than using hourly averaged data, the 10-minute average of measured environmental parameters was computed. This secondary data set served both to validate model performance results on the hourly averaged data set, as well as to investigate the importance of the measurement interval in generating accurate predictions of C_n^2 . To this end, a Davis VANTAGE PRO2 weather station was established approximately 4 m from the receiver of the scintillometer link between December 10th 2019 and March 31st 2020. This weather station measured atmospheric parameters at an elevation of approximately 1-2 m above the surface of the water, from the edge of the USNA Severn River Basin pier. Local wind speed, air temperature, relative humidity, atmospheric pressure, and rain measurements were collected. A current profiler, which measured the properties of the Severn River at a depth of approximately 4 m, was fixed to the below the local weather station. The current profiler was used to measure the water temperature and water depth. A description of the measurements collected with units of measure, time interval, and location is given in Table 3.

Table 3: Local Davis Weather Station Measurements with Units, Time Interval, and, Location [19].

Parameter	Measurement Interval	Units	Source:
Air Temperature	Observations reported at 2-4 minute intervals.	[°F, °C]	Davis VANTAGE PRO2 Weather Station Date Range: December 10 th 2019 to 31 st March 2020 Latitude: 38.9832 °N. Longitude: -76.4792 °W. Elevation from Ground: 2 ft. 
Relative Humidity	Observations reported at 2-4 minute intervals.	[bar]	
Dew Point Temperature	Observations reported at 2-4 minute intervals.	[bar]	
Wind Speed	Observations reported at 2-4 minute intervals using interval averaging.	$[\frac{m}{s}]$	
Atmospheric Pressure	Observations reported hourly intermittent averaging.	[hPa]	
Rainfall	Observations reported hourly, total rainfall for the duration reported.	[mm]	
Solar Radiation	Observations reported at 2-4 minute intervals using interval averaging.	$[\frac{W}{m^2}]$	
UV Index	Observations reported at 2-4 minute intervals. Derived from pyranometer measurements.	$[\frac{m}{s}]$	
Air Density	Observations reported at 2-4 minute intervals.	$[\frac{g}{m^3}]$	

In addition to the atmospheric parameters collected in Table 3, oceanographic and temporal data was collected. Water depth was measurements from the current profiler were reported from 4 m below the surface of the Severn River at high tide, to the actual river surface. This parameter was included in an attempt to capture local tidal and wave information. In the same manner as for the expansive hourly averaged data set, the scintillometer internal clock was used to calculate the temporal hour weight and seasonality for use in this higher-frequency local data set. These parameters were calculated using equations (8) and (9). Although the time period over which this data was collected was shorter than for the hourly averaged data set, the higher measurement frequency resulted in a similar number of total possible observations. Both data sets discussed were prepared and analyzed before model development.

3.2 Data set preparation

Before progressing to the final compilation and standardization of the time series data, the scintillometer measurements were cleaned to eliminate observations with missing values or non-physical measurements. For example, due to temporary obstructions such as water traffic or dense fog, some scintillometer readings were reported as missing. Additional scintillometer readings that were well above 1×10^{-12} were removed from the data set as non-physical. Local weather data collected over 15-minute intervals was averaged similarly in order to ensure consistent measurement of environmental parameters. Missing or non-physical measurements reported by the Severn River Watershed Observatory were eliminated using hourly averaging of the reported data, in a similar manner as the NOAA and NDBC stations [23, 24]. Both the NOAA and NDBC parameter measurements were obtained from their respective quality controlled data bases, where the readings were reported as hourly averages, with outliers and missing data handled by the NOAA or NDBC procedures before publication [23, 24]. When measurement from the local observatory was not available, the corresponding NOAA or NDBC measurement was used to complete the data set. Rather than using the water temperature as measured, the air water temperature difference was obtained by subtracting the measured water temperature from the measured air temperature for each observation.

These atmospheric, oceanographic, and temporal parameters had varying measurement intervals. In order to perform analysis on the data set, the time of each observation had to be standardized, so that for each index in the time series, a scintillometer reading had corresponding environmental parameter readings. An hourly time series of readings from January 4th 2019 until March 31st 2020 was generated using time synchronization. In order to bring the timing of all of the data sources into reasonable alignment, hourly data averages were computed. This allowed both a wide range of environmental parameters, as well as use of quality controlled NDBC and NOAA data. Between January 4th 2019 and March 31st 2020, there were 10,909 possible hourly observations. The final data set contained 5,133 complete observations during this time period. There were a number of short outages in data recording, between 3 hours and 3 days, due to scintillometer misalignment, snowfall, loss of power due to extended overcast days, as well as local boat traffic blockages across the propagation path in the Severn River. Two longer data outages occurred during May and July of 2019, attributed to scintillometer misalignment. As a

large number of data sources was used, loss of data from any one source contributed to the reduction in the number of observations in the data set. A custom MATLAB function was written to produce a single data set from the various data sources collected. This function is given in Appendix A. A sample of the data set containing C_n^2 and the 12 environmental parameter observations used in our analysis is provided in Table 4.

Table 4: Hourly Averaged C_n^2 and Environmental Parameter Observation Data Set.

Time	Cn2	Air Temperature [C]	Air-Water Temperature Difference [C]	Dew Point Temperature [F]	Pressure [bar]	Hourly Pressure Change [bar]	Relative Humidity [%]	Wind Direction [degT]	Gust Wind Speed [m/s]	Average Wind Speed [m/s]	Visibility [nm]	Seasonality	Temporal Hour Weight
1/2/2019 01:00	4.79E-16	6.60	-3.00	46	29.78	0.08	93	140	18	3	0.75	0	-0.68
1/2/2019 02:00	4.77E-16	6.60	-2.00	48	29.78	0.08	89	150	18	3	1	0	-0.57
1/2/2019 03:00	5.29E-16	6.60	-1.70	46	29.78	0.08	88	140	18	3	0.5	0	-0.46
1/2/2019 04:00	3.32E-16	6.60	-1.10	47	29.78	0.08	89	140	18	3	0.25	0	-0.36
1/2/2019 05:00	3.25E-16	6.60	-0.90	45	29.78	0.08	86	130	18	6	0.5	0	-0.25
1/2/2019 06:00	1.59E-15	6.60	-0.70	45	29.75	0.08	86	130	18	9	0.25	0	-0.15
1/2/2019 07:00	5.01E-15	6.60	0.00	46	29.75	0.08	90	130	18	5	0.5	0	-0.04
1/2/2019 08:00	3.18E-15	6.60	0.30	46	29.75	0.08	86	130	18	6	0.5	0	0.06
1/2/2019 09:00	2.81E-15	6.30	-0.10	48	29.75	0.08	89	220	18	6	0.5	0	0.17

In addition to the hourly averaged data set in Table 4, the secondary data set using only higher-frequency local environmental parameter measurements and local water temperature was developed. These readings were reported at 10 minute intervals between December 10th 2019 and March 31st 2020. Of the 15,624 possible 10-minute readings in this period, 10,253 were collected. The same short measurement outages in the scintillometer link affected the 10-minute averaged data set. However, no data outages longer than 4 days occurred during the observation period. MATLAB was used to compile observations from each data source for the local 10-minute averaged data set; this function is given in Appendix B. A portion of the 12 environmental parameter data set used in our analysis follows in Table 5.

Table 5: 10-Minute Averaged C_n^2 and Environmental Parameter Observation Data Set.

Time	C_n^2	Air Temperature [C]	Relative Humidity [%]	Dew Point Temperature [F]	Wind Speed [m/s]	Pressure [hPa]	Rainfall [cm]	Solar Radiation [W/m ²]	UV Index	Air Density [kg/m ³]	Depth [m]	Temporal Hour Weight	Air-Water Temperature Difference [C]
12/10/2019 05:10	5.41E-14	20.9	54	11.2	0	1011.6	0	69	0	1.18	4.99	-0.22	10.48
12/10/2019 05:20	5.57E-14	20.8	54	11.2	0	1011.7	0	51	0	1.18	4.99	-0.20	10.38
12/10/2019 05:30	5.65E-14	20.8	54	11.2	0	1011.5	0	0	0	1.18	4.99	-0.18	10.38
12/10/2019 05:40	5.69E-14	20.9	54	11.2	0	1012	0	22	0	1.18	4.99	-0.16	10.48
12/10/2019 05:50	5.68E-14	20.9	54	11.3	0	1012	0	44	0	1.18	4.99	-0.15	10.48
12/10/2019 06:00	5.74E-14	20.9	54	11.3	0	1011.9	0	44	0	1.18	4.99	-0.13	10.48
12/10/2019 06:10	5.76E-14	21.1	54	11.4	0	1012	0	44	0	1.18	4.99	-0.11	10.68
12/10/2019 06:20	6.66E-14	21.1	55	11.7	0	1011.9	0	44	0	1.18	4.99	-0.09	10.68
12/10/2019 06:30	7.11E-14	21.2	55	11.8	0	1011.4	0	44	0	1.18	4.99	-0.08	10.78

After developing the hourly averaged and 10-minute averaged data sets in Tables 3 and 4, the measurements of observed C_n^2 and the corresponding environmental parameters were explored. This was done in an effort to validate the measurements as reasonable and in line with expectations set out in prior literature [1, 2, 3, 8, 10, 11, 12].

3.3 Data Set Overview

After cleaning, standardizing, and compiling, the environmental parameter readings and C_n^2 measurements from January 2019 to March 31st 2020, the ranges and distribution of measurements were analyzed. Values for C_n^2 measured during this time ranged from 2.23×10^{-16} to 3.96×10^{-12} . For applications in laser communication and weapon systems, C_n^2 values from 1×10^{-16} to 5×10^{-12} are generally expected. The field measurements of C_n^2 along the propagation path were partitioned by month, and the resulting observation sets plotted as histograms. Both the range and distribution of C_n^2 readings were analyzed and compared to existing literature. Monthly histograms of readings from January 2019 to March 2020 are given in Figure 5.

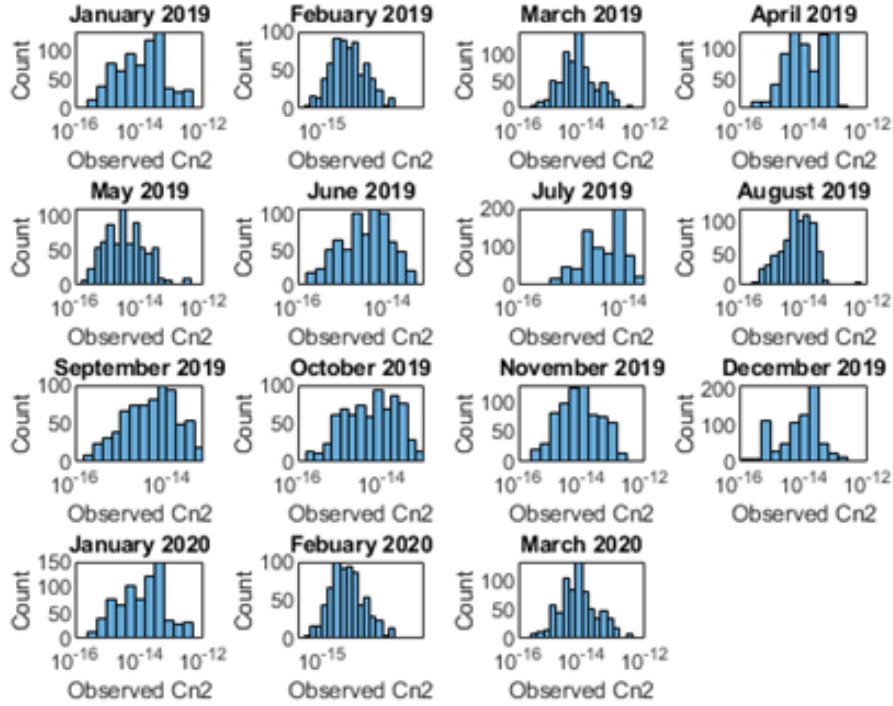
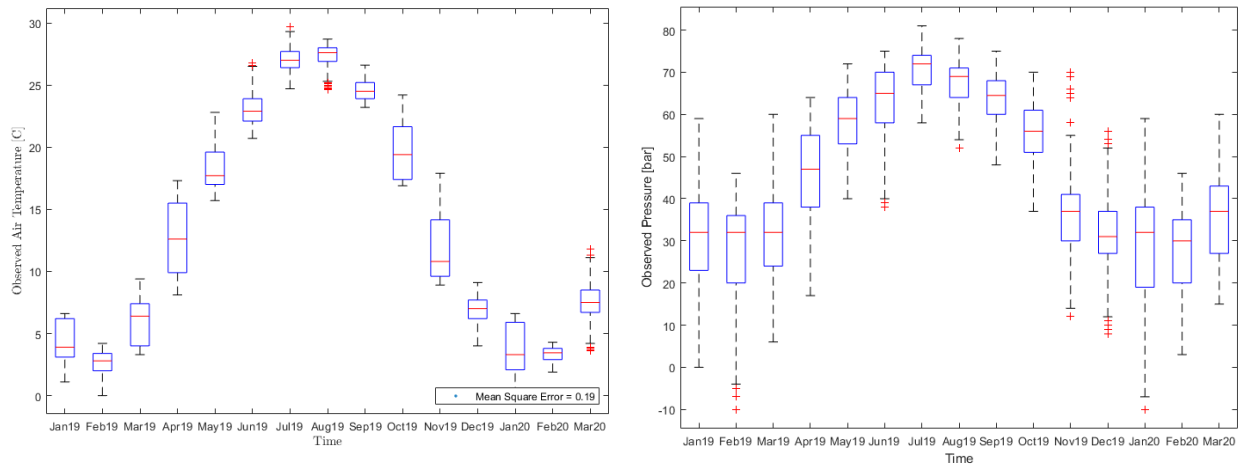


Figure 5: Monthly Histograms of Observed C_n^2 From January 2019 Through March 2020.

Both the range of C_n^2 values observed and their distribution are similar to the distributions presented in literature [2, 10, 12]. The distributions of C_n^2 showed a unimodal distribution with fat tails similar to those presented in literature [10].

The ranges and distributions of the measured hourly averaged environmental parameters were also examined to ensure they fell within the bounds of existing literature. The recorded observations for the full 15-month period are given as bar and whisker plots in Figure 6.



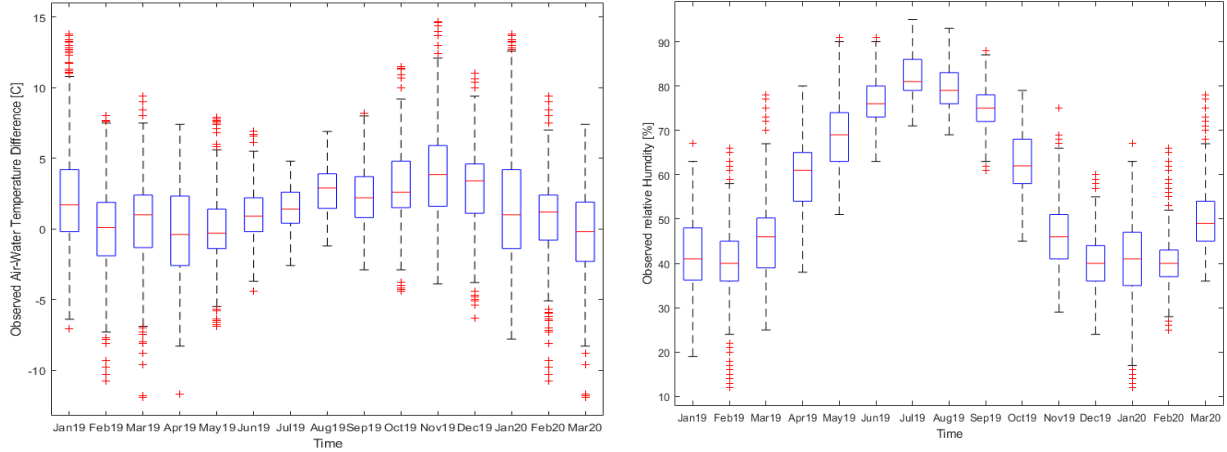
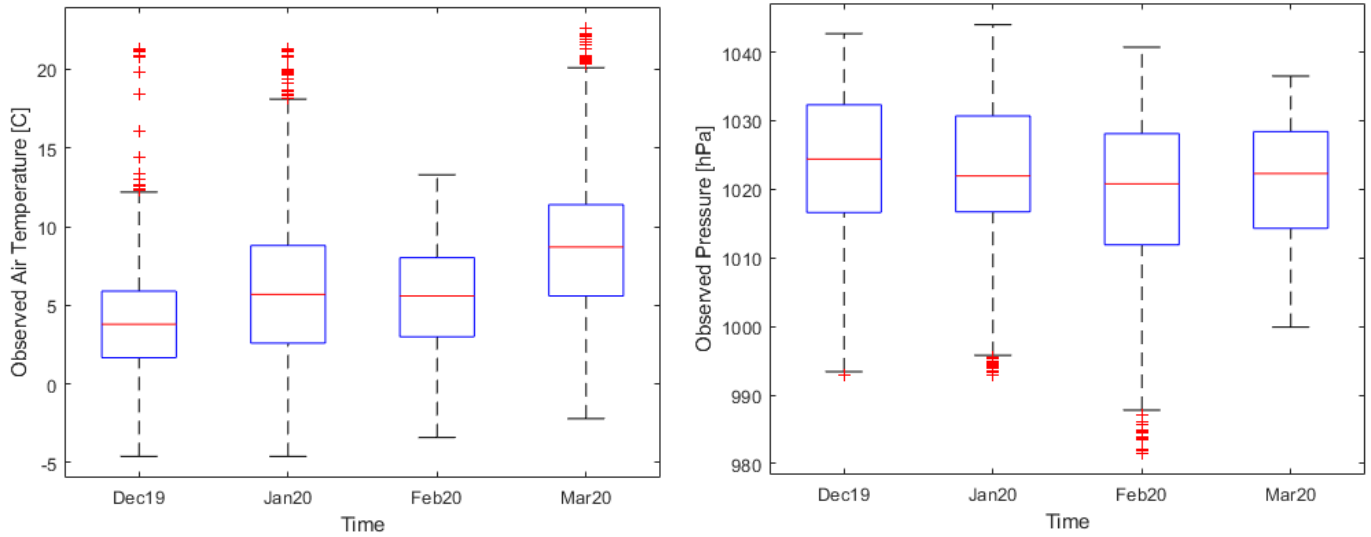


Figure 6: Bar and Whisker Plots of Observations of Selected Environmental Parameters (Hour-Averaged).

As with C_n^2 , both the range and distribution of environmental parameter measurements are similar to the distributions presented in literature [3, 8, 10, 12]. After this validation of the hourly averaged data set, the high-frequency local data was analyzed in a similar way. The ranges and distributions of the 10 minute interval readings in each calendar month are presented as bar and whisker plots in Figure 7.



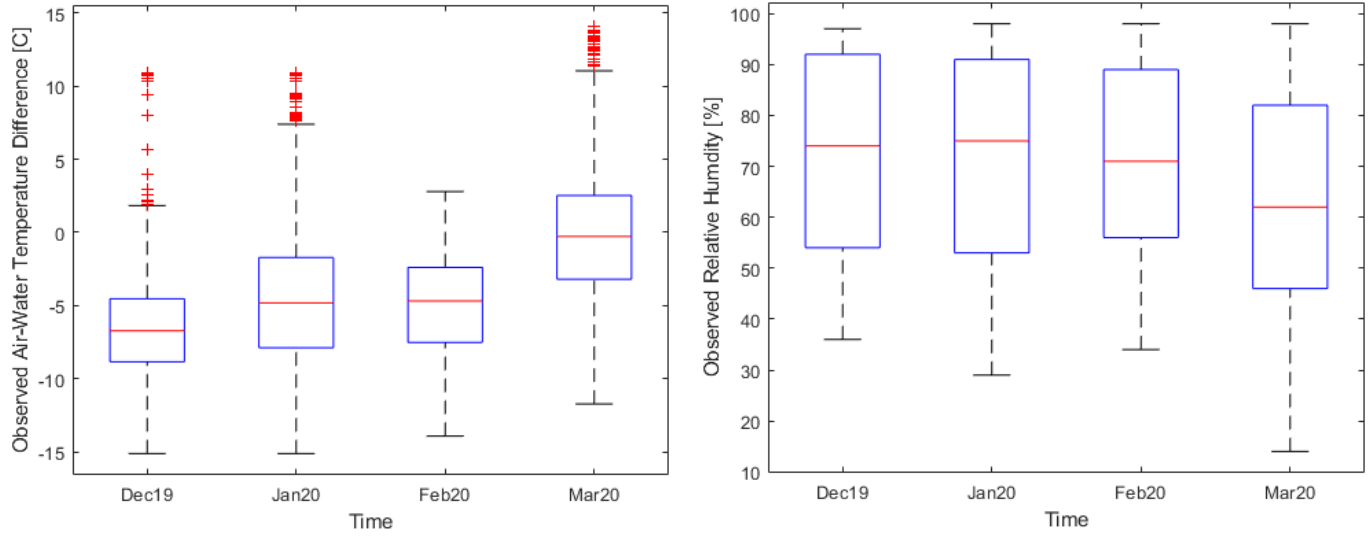


Figure 7: Bar and Whisker Plots of Observations of Selected 10-minute Averaged Environmental Parameters.

The 10-minute averaged data set also contained measurements in line with expectations from other literature, and the data in the hourly averaged set. After validating the environmental parameters and C_n^2 observations which form the core of the data sets, a variety of models were trained.

4 Modeling

After exploratory data analysis and validation of scintillometer readings, machine learning techniques were applied to generate new understanding of potential physical relationships between environmental parameters and optical turbulence. Both the hourly year-long data set and the 10-minute averaged local data set were used to train, develop, test, and investigate candidate models.

The primary goal when training machine learning models using the two data sets described in the prior section was to generate predictions for the structure parameter of the refractive index C_n^2 . To this end, C_n^2 was taken as the *response variable* for model development. Predictions for the response variable C_n^2 were made using the measured local environmental parameters, called *predictors*.

A sub-set of 60% of these observations was randomly selected to train machine learning models. A further 20% was set aside as a development set, allowing these models to be tuned to optimize performance. The final 20% of the data set was set aside as a test set. This allowed validation of the model on data that was never used in training.

The data sets used for training and testing contained observations of C_n^2 and the corresponding measurements of 12 distinct environmental parameters. For model development and evaluation, C_n^2 was selected as the response variable, denoted $y^{[i]}$ when referencing an individual observation i , or denoted $\mathbf{y}$ when denoting the column vector of all observations. Each of the environmental parameter column vectors were taken as corresponding predictor variables, $\mathbf{x}_f$ for $f \in \{1, 2, \dots, 12\}$. The vectors $\mathbf{y}$ and $\mathbf{x}_f$ each contained n observations, indexed by $i \in \{1, \dots, n\}$. The use of indexing allows the modeling approach to generate models using either the hourly averaged or the 10-minute averaged data set. Thus, for each observation at a given time for the training, development, and test sets, the data matrix has a row of the form:

$$[y^{[i]} | \mathbf{X}^{[i]}] = [y^{[i]} | x_1^{[i]}, x_2^{[i]}, x_3^{[i]}, \dots, x_{12}^{[i]}]$$

for each observation $i \in \{1, \dots, n\}$. In the hourly averaged data set, the 5,133 observations were split into a training set of 3,080 observations selected randomly without replacement. The development set contained 1,027 observations, also selected randomly without replacement from the set of observations not used in training. The remaining 1,026 observations were selected for the test set. Each of these sets contained 13 columns, one for the response C_n^2 and 12 for the measured environmental parameters.

Similarly, in the 10-minute averaged data set, the 10,253 observations were split into a training set of 6,152 observations selected randomly without replacement. The development set contained 2,051 observations, also selected randomly without replacement from the set of observations not used in training. The remaining 2,050 observations were selected for the test set. In the same manner as the hourly averaged data set, there were 13 data value columns, one for the response C_n^2 and 12 for the measured environmental parameters.

Using these two data sets, three different models were trained, developed, and tested. First, a general linear model was trained on each data set to determine the potential efficacy of parametric linear models in generating predictions for C_n^2 . The training and testing of general linear models is explored in section 4.1. These models' performance was compared to existing literature, which use linear models readily for C_n^2 prediction [2, 3, 5, 6, 8, 10]. Next, regression trees were trained and tested, with the full process described in section 4.2. Two additional sets of models which used regression trees as a base component were trained in sections 4.3 and 4.4.

In order to train the regression tree based models, a variety of potential values for *hyper-parameters* were evaluated. Hyper-parameters are constants built into machine learning models which cannot be learned from a given data set. Their value is established prior to model training. In the case of regression trees, the structure of each individual tree was in part dictated by the established values for hyper-parameters. As the optimal value cannot be learned directly from the data set, a process called *tuning* is used to determine appropriate values before the final model is trained. Tuning involves training regression trees with a wide variety of combinations of hyper parameters, with literature suggesting a range of appropriate values for each [13, 14, 27, 28, 29]. Models are then trained using each combination of possible hyper-parameters and compared with one another using the development set. The most appropriate hyper-parameter combination from the tuning phase is used to train a final model, using a combination of the training and development data sets. The final model is then evaluated using the observations in the test set. Each of these approaches, the trained model, and the model effectiveness will be investigated in turn.

4.1 General Linear Model

Current Linear models are widely applied in optics for C_n^2 prediction. The structure of a linear model allows for prediction of C_n^2 as a response variable from continuous structure parameters or environmental measurements. A similar approach, which used a machine learning informed iterative training process called the stepwise general linear model was developed and compared to the prediction accuracy results suggested by this literature [2, 3, 8, 10, 11, 12]. This approach differs from standard linear model fitting in that it uses a training set, development set, and test set to train the model. The number of terms in the model is adjusted according to the reduction in development error from each term.

Using the environmental parameter data matrix $\mathbf{X}$ in the training set, a general linear model is trained to produce a prediction vector for C_n^2 , denoted by $\hat{\mathbf{y}}$. This model is of the form in equation (12).

$$\hat{y} = \beta_0 + \sum_{j=1}^{12} \beta_j \mathbf{X}_j \quad (12)$$

In equation (12), the 12 environmental parameters indexed by at most $j = \{1, \dots, 12\}$. If an environmental parameter in this set does not reduce model error, it is removed from the final model. These environmental measurement data columns $\mathbf{x}_j$ are multiplied by their corresponding model coefficient β_j . Some literature denotes these coefficients using $\hat{\beta}$ to indicate that the value for β is estimated by the model. In order to determine appropriate values

for these β_j , the least squares approach seeks to minimize a given *objective function*. The most common objective function, and the one used in this study, is the residual sum of squares (RSS), also called the sum of square errors (SSE). Typically, β_0 is incorporated into the vector of coefficients $\boldsymbol{\beta}$, and a constant 1 is included in the vector of predictor variables at each observation. The values for each β_j in the vector of coefficient $\boldsymbol{\beta}$ are determined by minimizing the SSE objective function in equation (13).

$$\min(SS_E(\boldsymbol{\beta})) = \min \left(\sum_{i=1}^n y^{[i]} - \mathbf{x}^{[i]'} \boldsymbol{\beta} \right)^2 \quad (13)$$

In equation (13), the vector of coefficients $\boldsymbol{\beta}$ are selected to minimize the square difference between a prediction for C_n^2 and the corresponding observation for C_n^2 for the n observations indexed by i in the training data set. The linear model in equation (12) includes one coefficient for each environmental parameter as well as a baseline parameter β_0 . This results in a fixed number of model parameters which does not directly model interactions between environmental parameters. An expansion of the linear model is the general linear model approach, which allows the number of parameters in the model to vary within a fixed range.

The general linear model allows for a range of possible model parameters, and does not assume a direct linear relationship between C_n^2 and any one environmental parameter. Rather, it assumes that each environmental parameter is normally and independently distributed, and that there is a linear relationship between a transformation of C_n^2 and a linear combination of environmental parameters.

As the physical interactions which affect optical turbulence are not fully understood in the near-maritime environment, interaction terms were included in the linear model. Interaction terms seek to capture effects from the combination of multiple environmental parameters simultaneously. Rather than limiting the general linear model to one term for each environmental parameter predictor, the model can be allowed to fit coefficients for interaction parameters. For two predictors $\mathbf{x}_j$ and $\mathbf{x}_k$, a basic linear model will have some coefficients β_j and β_k used to scale observed measurements. If interaction terms are allowed, a new predictor $\mathbf{x}_{jk}$ is defined by $\mathbf{x}_{jk} = \mathbf{x}_j \cdot \mathbf{x}_k$. This interaction parameter also has a corresponding coefficient β_{jk} . In order to train an appropriate general linear model, the training set for the hourly averaged and 10-minute averaged data were used to estimate values for the coefficients β_i . First, a linear model with no interaction terms was fit to the training data which sought to minimize the mean square error on the development set. Then, a second model which allowed interaction terms was trained and evaluated. The modeling structure is further explored in [30, 31].

4.1.1 Training

The stepwise regression method is used to add and remove terms from the starting linear model based on the estimated statistical significance of each term in predicting C_n^2 . The starting model in all cases was a standard linear model in which each of the 12 predictors were included, with corresponding coefficients $\beta_1, \dots, \beta_{12}$. In the first trial, no interaction terms were allowed, and the stepwise regression method sought to remove unnecessary parameters after fitting appropriate values for each β_i . In order to determine whether to include or exclude a term from

the model, a p-value of 0.05 was set as a tolerance. When a predictor had an estimated p-value below the tolerance, it was included in the next iteration of the model. If a parameter had an estimated p-value greater than the tolerance, it was removed in the next iteration of the model. When no parameters excluded had a p-value below the tolerance, and no parameters included had a p-value above the tolerance, the process ended. The model at the conclusion of the stepwise regression process was taken as the final model, and used to predict C_n^2 for the test set in the hourly and 10-minute averaged cases. The MATLAB script used to train the general linear model in the hourly averaged case is given in Appendix C, while the 10-minute average training script is given in Appendix D.

4.1.2 Results and Discussion

The model resulting from the stepwise training process for the model with no interaction terms allowed is given in Figure 8.

Linear Model With no Interaction Terms, Hourly Averaged Data Set				
$C_n^2 \sim \beta_0 + (Air\ Temp)\beta_1 + (Air - Water\ Temp)\beta_2 + (Rel.\ Hum.)\beta_3 + (Visibility)\beta_4 + (Seasonality)\beta_5 + (W_{th})\beta_6$				
Coefficient		Estimate	Square Error	p-value
	β_0	-14.01	0.054	~ 0
	β_1	-0.013	0.001	1.16×10^{-48}
	β_2	0.033	0.002	8.76×10^{-48}
	β_3	0.002	0.001	1.21×10^{-5}
	β_4	0.008	0.0034	0.0176
	β_5	-0.071	0.026	0.005481
	β_6	-0.246	0.024	7.08×10^{-28}

Figure 8: Stepwise-GLM Training Process for Hourly Averaged Training and Development Data Sets.

In addition to the stepwise linear model developed in Figure 8, a second model which allowed interaction terms between the 12 environmental parameters in the data set was trained. In both cases, the observations excluded from training in the test set were used to evaluate model performance. In order to measure model performance, predicted C_n^2 was compared with observed C_n^2 using the test set. Observations in this set were never trained by the model, in order to more accurately reflect model performance. Because of the low order of magnitude in C_n^2 observations, the base 10 logarithm of both the predicted C_n^2 and the observed C_n^2 were taken before calculating model error. The error metric used in this study is the *mean square error* (MSE) obtained using equation (14).

$$MSE = \frac{1}{N} \sum_{i=1}^N (y^{[i]} - \hat{y}^{[i]})^2 \quad (14)$$

In equation (14), the general linear model prediction $\hat{y}$ for an observation i in the test set, which contains N total observations, is compared to the observed value of C_n^2 denoted by y . The model mean square error for predicting $\log_{10} C_n^2$ is the key metric for comparing model efficacy. A model which predicted the mean observed value for $\log_{10} C_n^2$ would have an MSE around 0.35

for either the hourly averaged or 10-minute averaged data set. Some current explicit models when applied in over water environment can demonstrate typical MSE between 0.25 and 0.10 [31]. Scatter-plots of observed $\text{Log}_{10}C_n^2$ and corresponding linear model predictions for $\text{Log}_{10}C_n^2$ are shown below for both the hourly and 10-minute averaged data sets, using both the linear and interaction models. Figure 9 lists the linear and interaction model performance on the hourly averaged data set on the top row, and the corresponding performance for the 10-minute averaged data set below.

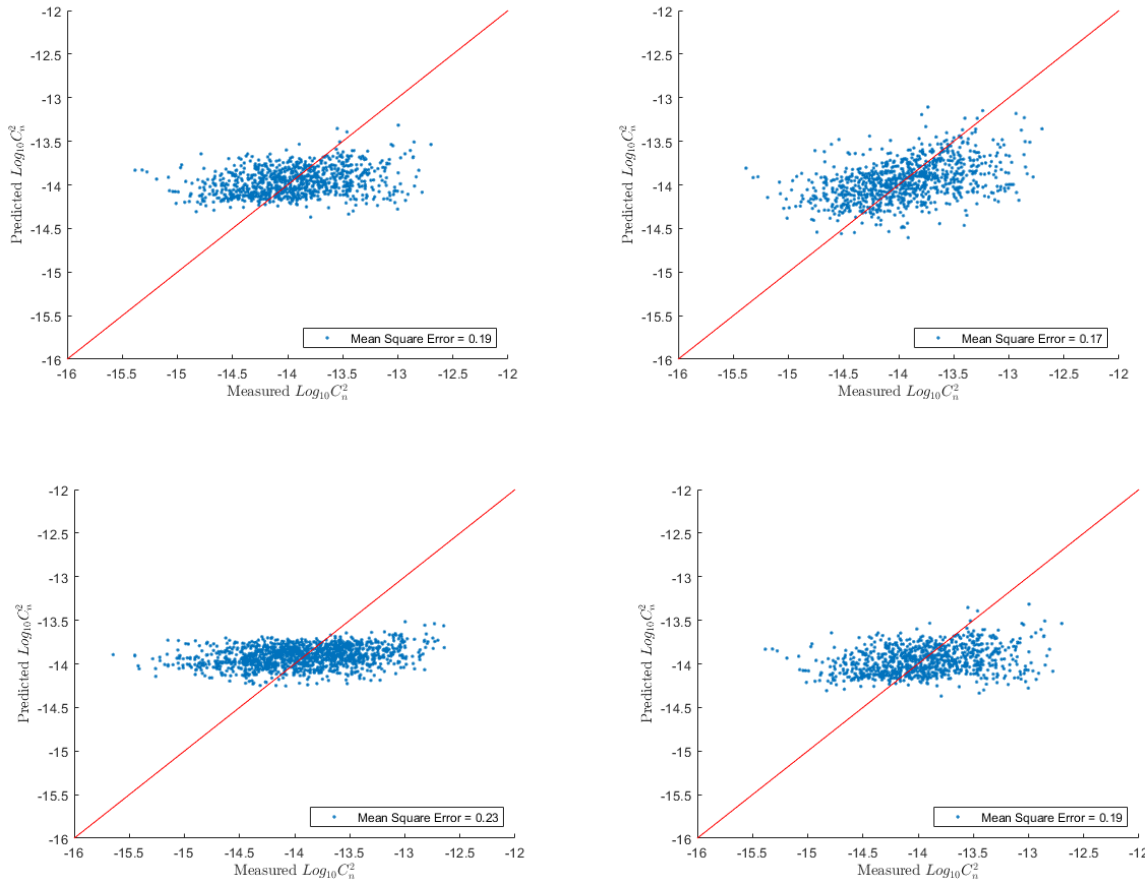


Figure 9: Hourly averaged Linear (Top-left) and Interactions (Top-right), as well as 10-minute Averaged Linear (Bottom-left) Linear and Interactions (Bottom-right) Model Prediction Accuracy.

Based on the results of model testing in Figure 9, a general linear model trained using the stepwise development approach had slightly larger error than expected from some current literature [2, 3, 8, 32]. However, as existing linear models often struggle to accurately predict C_n^2 in the near-maritime environment, the performance is in line with expectations [3, 8, 32]. Notably, in both the hourly averaged and 10-minute averaged data sets, the inclusion of interaction parameters improved model test accuracy. The reduction in error due to interaction terms is indicative of the complexity of the relationships between environmental parameters and optical turbulence. Interaction parameters between air temperature and air-water temperature difference, and between air temperature and pressure, were the main sources of error reduction.

Even with the reduced error provided by interaction parameters, all general linear models trained failed to accurately capture the physics behind optical turbulence. Both the distribution of residuals and the failure to accurately predict extremes of C_n^2 are visible in Figure 9. These general linear models established a baseline goal of model mean square error below 0.2 for the prediction of $\text{Log}_{10} C_n^2$ in both the hourly and 10-minute averaged data sets.

After investigating linear models' fit using the stepwise development approach, nonparametric models were investigated. These models did not establish a fixed number of model parameters before training, rather the training data was used to determine an effective number of model terms. Regression trees were the first class of data-driven machine learning model investigated.

4.2 Regression Trees

A regression tree is a nonparametric model which seeks to divide the data set into a number of distinct, non-overlapping regions in order to generate a prediction for a response variable given an input predictor observation. In this paper, regression trees will seek to partition $\mathbb{R}^{12}$, such that given some observation $\mathbf{x}^{[i]} = x_1^{[i]}, x_2^{[i]}, x_3^{[i]}, \dots, x_{12}^{[i]}$ excluded from the training set, there exists a prediction $\hat{y}^{[i]}$ that depends only on the region in which $\mathbf{x}^{[i]}$ lies.

As a model, each regression tree takes the form of a nonparametric model from $\mathbb{R}^{12}$ to $\mathbb{R}$. While a standard linear model on $\mathbb{R}^{12}$ is smooth, a regression tree surface can be jagged [29]. This may enable the regression tree model to better capture the physics associated with near-maritime optical turbulence. Using these definitions, the algorithm used to train a regression tree from a given set of training data will be explored. One key advantage of regression trees as models is that they are *consistent*. A model is consistent if its prediction accuracy increases as the size of the training set increases [33]. Consistency in regression trees is key to their effectiveness as predictive tools. It is essential both to the validity of predictions generated using individual regression trees and to models which use them as a component, that larger volumes of training observations do not begin to generate worse predictions. As with other classes of machine learning models, regression trees can be shown to be consistent [13, 14, 16, 17]. Due to model consistency, prediction accuracy improves as the set of observations used to train the tree grows, provided it is representative of the sample population.

4.2.1 Training

In order to train a regression tree model, the training set consisting of 60% of total observations is used. This data set becomes the first node in the regression tree. In the context of regression trees, a *node* is a collection of observations at some level in the tree. To begin, all observations are in the same node at the top of the regression tree. In order to generate lower levels in the tree, the initial parent node is split into two child nodes [13, 14, 16, 29, 34]. We select one variable and a value, and separate the observations into two groups based on whether the observation is higher or lower than that value. In order to identify the location of a split within an environmental parameter, a *splitting criterion* is used [13, 29]. The standard splitting criterion is the SSE objective function previously discussed. The sum of squared differences between $\hat{y}$ and each observed y in a set of two candidate nodes is compared to determine which nodal split is most appropriate. While a split could, in theory, happen anywhere between two observations,

many potential splits result in the same set of child nodes. As there are no more than 10,253 observations and 12 environmental parameters in the largest data set, the total number of unique splits at each decision point is at most 123,056. All unique splits within each of the 12 predictor variables are compared. The potential split which minimizes the SSE for a given node becomes the nodal split, which divides the node into two smaller, non-overlapping child nodes whose union is the parent node.

This process is repeated iteratively, in a process called recursive binary splitting. Because each node is divided into exactly two child nodes, each split is *binary*. This process happens iteratively until some stopping criterion is reached. Typically, a minimum number of observations is set as the stopping criterion, such that each child node has at least a given number of observations. When further splitting will generate new nodes with fewer than the required number of observations, the node becomes a terminal node. Thus, every possible observation will fall into exactly one terminal node in the regression tree.

The process for developing a regression tree from a sample training data set is further developed using the algorithm in Figure 10 [15].

1. *Begin with a top – level node containing all training observations.*
2. *Set a stopping criterion, such as a minimum number of observations required for each node.*
3. *Analyze all potential splits within each predictor of observations in the node until the split which minimizes the sum of square error is identified.*
4. *Take this maximal split as the nodal partition, dividing the node into two child – nodes along this split in feature space.*
5. *Repeat 2 – 3 within each child node until the stopping criterion is reached.*

Figure 10: Algorithm Used to Train Regression Tree Model Using Recursive Binary Splitting [15].

Using Figure 10, a basic regression tree was trained which sought to split the training data sets into two terminal nodes. Splitting the training data set into two terminal nodes involved finding the maximal split among all environmental parameters. As in Figure 10, the maximal split was identified by comparing the sum of square error for all potential splits. This model is a *stump* regression tree. A stump regression tree is the simplest regression tree model investigated, and training it involves only making the single most effective split.

4.2.2 Results and Discussion

The stump regression trees for the hourly averaged and 10-minute averaged data were each trained as a starting point for machine learning based model analysis. For the hourly averaged data set, the maximal split divided the training data based on if the air-water temperature difference was above or below 4.7 [°C]. In the 10-minute averaged training data set, the maximal split divided observations with air density below $1.28 \left[\frac{g}{m^3} \right]$ with observations above this value. When an observation had measured air density below the splitting value, the model predicted C_n^2 to be $1 \times 10^{-13.981}$, otherwise the model predicted C_n^2 to be $1 \times 10^{-13.66}$. This basic process

could be repeated to further split the two terminal nodes depicted in Figure 11 into many smaller terminal nodes, but the baseline estimate of model error for one split is useful in evaluating the relative performance of regression tree based models with higher complexity.

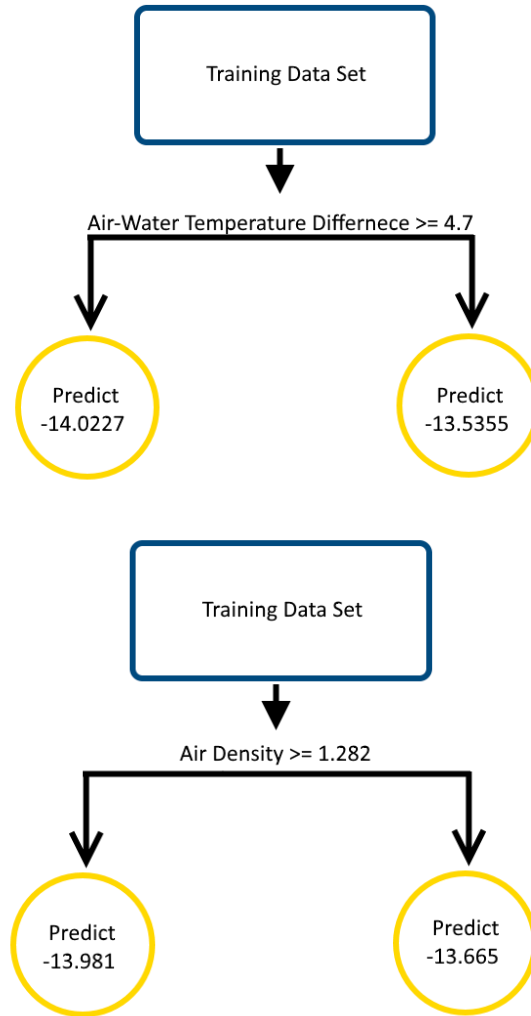


Figure 11: Stump Regression Trees for Hourly Averaged (Above) and 10-minute Averaged (Below) Data Sets.

In Figure 11, the stump regression tree for the hourly averaged data had a mean square error of 0.26 for predicating $\text{Log}_{10}C_n^2$. The 10-minute averaged stump regression tree had a slightly lower mean square error of 0.23. These metrics offer a baseline for which to compare regression trees with a higher number of splits, as well as models made up of a large number of individual regression trees.

In addition to investigating the stump regression tree, a *deep* regression tree was trained and evaluated. A deep regression tree seeks to split the training data set such that each observation in the training set is in its own terminal node. Thus, there is a unique prediction for C_n^2 denoted $\hat{y}^{[i]}$ for each observation of environmental parameters $\mathbf{X}^{[i]}$ in the training set. The deep

regression tree is the most complex regression tree for a given data set, in contrast to the single split stump tree. The total number of splits in a regression tree model can be tuned to optimize prediction accuracy. Basic tuning using both the 10-minute averaged and hourly averaged data sets indicated that deeper trees had lower prediction error than trees with fewer splits. The deep regression trees were used both to analyze prediction error and as a baseline for comparison with ensemble models. A full depiction of the deep regression tree is omitted, but a sample tree with 12 total splits is presented in Figure 12.

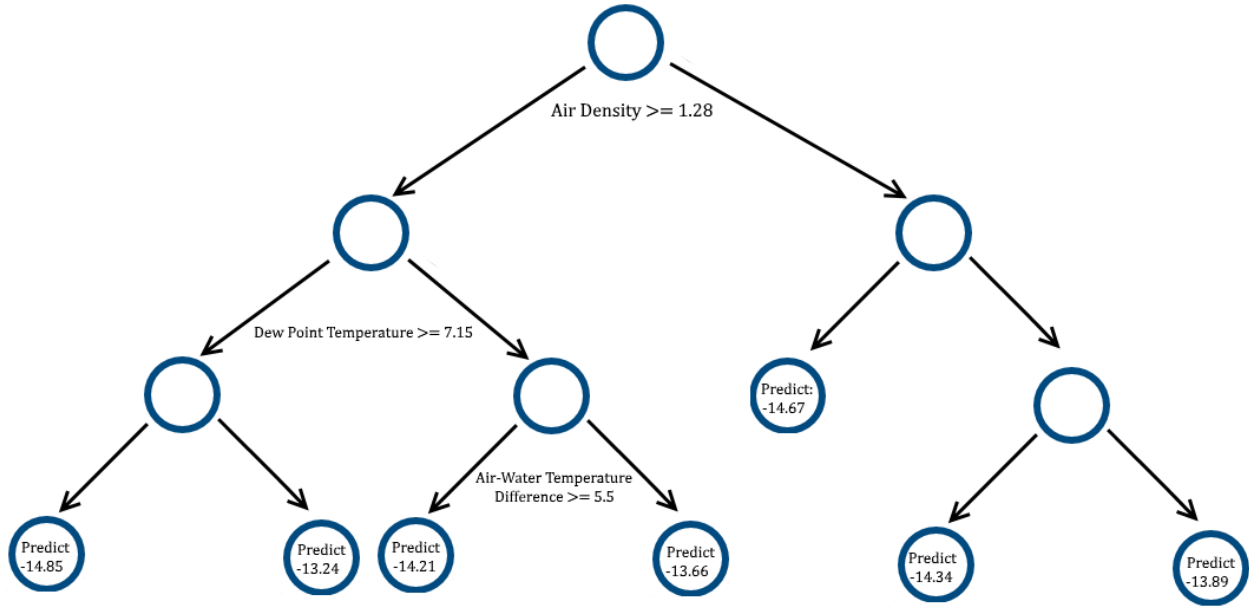


Figure 12: Sample Regression Tree with 12 Total Nodal Splits Trained on the 10-minute Averaged Data Set.

The regression tree in Figure 12 lies between a stump tree and a deep tree in complexity. In order to analyze the performance of the deep regression tree, it can be trained on the full training and development set, seeking to split each node until only one observation exists in each terminal node. Starting at the top-level node which contains all training observations, recursive binary splitting is applied to split the data set into smaller nodes. In order to generate a predicted value for $\text{Log}_{10}C_n^2$ using the model in Figure 12, the observed environmental parameter measurements can be used to follow splitting criterion from the top-level node through to the terminal node which the observed environmental measurements fall into. The model's prediction for $\text{Log}_{10}C_n^2$ is listed below each terminal node. This prediction is obtained by averaging the observed $\text{Log}_{10}C_n^2$ for training data which falls into each terminal node. The model's performance is analyzed by comparing the predicted C_n^2 for an observation of environmental parameter measurements with the corresponding observed C_n^2 in the test set in Figure 13.

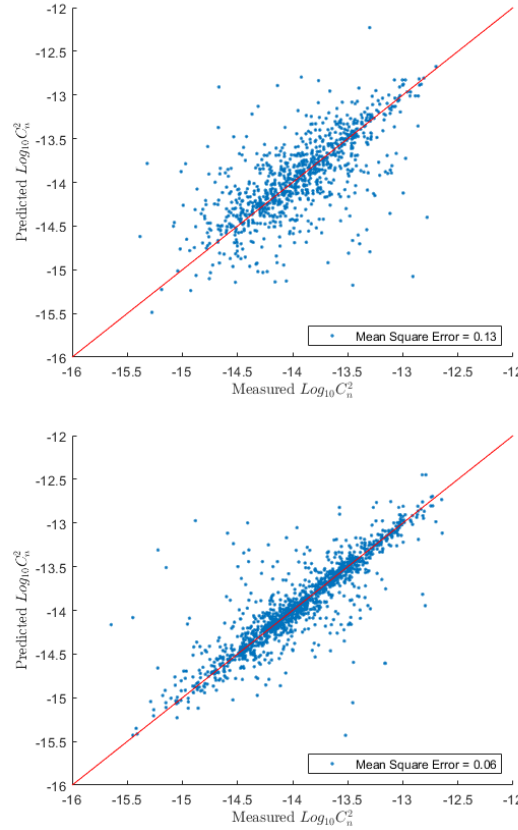


Figure 13: Hourly Averaged (Top) and 10-minute Averaged (Bottom) Deep Regression Tree Prediction Accuracy.

The prediction accuracy for the stump and deep regression trees demonstrates the potential advantage of non-parametric models for capturing the complex interactions between environmental parameters and optical turbulence in the near-maritime environment. When compared to optimized general linear models, even those including interaction parameters, regression trees had lower prediction error. The reduction in MSE of 0.13 is notable especially in the 10-minute averaged case. While the optimized general linear model had a test MSE of 0.19 on this data set, the deep regression tree model achieved a test MSE of 0.06. This value outperforms some of the models investigated in literature [32]. The regression tree approach is often used as a component of larger, potentially more robust models [13, 14, 29].

While regression trees are relatively simple to generate and understand, they suffer from two key drawbacks. If the distribution of observations in $\mathbf{y}$ is not representative, regression trees can generate inaccurate predictions [13, 29]. One key way to address this issue is to ensure the data set used for training is large enough and the minimum number of observations in each terminal node is appropriate [13, 29]. Further, due to variability in the observations selected for training, two regression trees trained on different subsets of the same global training set can be very different [29]. This process can be made more robust by increasing the number of distinct regression trees examined, to form *ensemble* models [13, 28, 29]. Ensemble models leverage a larger number of distinct regression trees to generate predictions for the response variable. The

manner in which these individual regression trees are incorporated into a final model defines the boosted regression tree approach and the random forest approach. In the random forest approach, each regression tree trained is weighted equally, and the random forest prediction is the average of the predictions of each constituent tree [13, 29]. For the boosted regression tree, new regression trees have their weight scaled by a hyper-parameter called the *learning rate*, which sets the rate at which new trees are incorporated to the final model [13]. The learning rate is set between 0 and 1 and acts as a scaling factor to each new tree incorporated into the model. A starting regression tree is trained, and successive regression trees are first weighted by the selected learning rate, and then added to the model. In this way, the starting tree has more weight in the final model prediction than the following trees added to the model.

For the following analysis, the regression tree approach is used to develop larger random forest models and boosted regression trees. Trees are structured as in Figure 10, but with two stopping criterion in their training. The two stopping criterion selected are the maximum number of splits and the minimum number of observations in each node. The end condition which stops this iterative splitting process is a required minimum number of observations in each node.

4.3 Random Forest

While a regression tree can act as a full model, generating predictions for any input test observation, it is more commonly used as the building block for a larger ensemble machine learning model. One key challenge with applying individual regression trees as models is the high variance between regression trees trained on different random samples of the same test population. By generating an ensemble of individual regression trees, the overall variance in the model is reduced [13, 28, 29]. Further, as each tree is unbiased and has the potential to capture the physical interactions hidden in the data set, regression trees are valuable as components of larger models [13, 14, 29].

Ensemble models such as random forests generate predictions by leveraging the predictions of their constituent components. In the case of regression tree ensembles, a number of distinct regression trees are trained on a data set, and some weighting scheme is applied to the predictions of each tree in the ensemble. The ensemble model prediction is thus informed by the predictions of more than one regression tree, in an effort to maintain low model bias but reduce the high variance of a single tree. A visual description of how individual regression trees are combined into a random forest model is given in Figure 14.

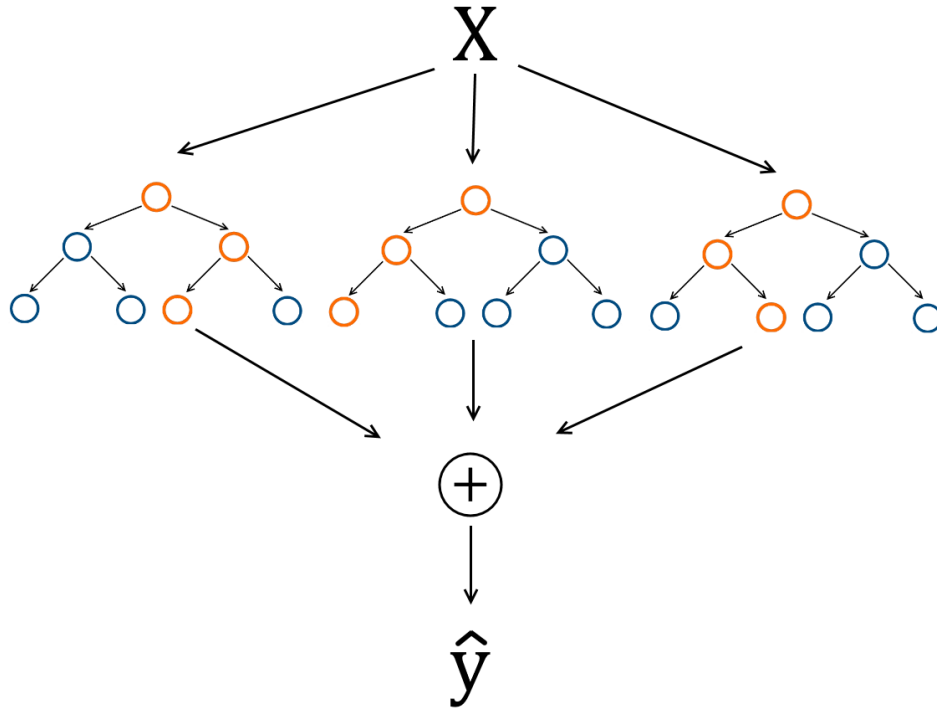


Figure 14: Developing a Random Forest from Individual Regression Trees.

In Figure 14, an observation $x^{[i]}$ is fed through many distinct regression trees, and each of the M regression trees denoted by T has their prediction $\hat{y}_T^{[i]}$ incorporated to the final random forest prediction $\hat{y}^{[i]}$ using equation (15).

$$\hat{y}^{[i]} = \frac{1}{M} \sum_{T=1}^M (y^{[i]} - \hat{y}_T^{[i]})^2 \quad (15)$$

The specific algorithm used to train each distinct regression tree, as well as the final random forest model, is developed in greater detail in the following section.

4.3.1 Training

The random forest ensemble approach generates many diverse regression trees which are incorporated into one model by averaging the prediction of each constituent regression tree [13, 14, 29]. The random forest approach induces diversity in individual regression trees by training each tree on a random sample of the entire data set. Additionally, each split in each tree is limited to a smaller, random subset of feature space in the training sample. Of the twelve features in a training data set, at most 9 were available for splitting. As with a standard regression tree, each split is made with the same goal of minimizing the SSE objective function. As the potential splits are limited to a random sample of all possible environmental parameters, no two regression trees are allowed to make the same series of splits.

The process used to train a random forest consisting of 250 regression trees, with 9 of 12 predictors randomly sampled as splitting candidates for each split is given in Figure 15 [29].

1. Select a random sample $\mathbf{X}^{sample}$ with replacement of size M from the training set.
2. Use this sample $\mathbf{X}^{sample}$ as the top level parent node; apply the recursive binary splitting regression tree fitting approach highlighted in Figure 10.
 - a. Randomly select 9 predictors as splitting candidates.
 - b. Determine the best split within the candidate predictors.
 - c. Take this best split as the partition, and generate two child nodes.
 - d. Continue applying recursive binary splitting until the stop condition of a certain minimum number of observations is reached.
3. Repeat step (2) until 250 regression trees, $T_1, \dots, T_{250}$ are fully trained.
4. Generate the random forest prediction $\hat{\mathbf{y}}$ using observations $\mathbf{X}$ in $\hat{f}(\mathbf{X})$ as:

$$\hat{\mathbf{y}} = \hat{f}(\mathbf{X}) = \frac{1}{250} \sum_{i=1}^{250} T_i(\mathbf{X})$$

Figure 15: Algorithm Used to Train the Random Forest Model [13, 29].

As each regression tree is likely to be unique, given that both the training data and parameters available for splitting at each node are randomized, the random forest model can provide a low-biased estimate of observed optical turbulence $y^{[i]}$ based on the corresponding environmental parameters in $\mathbf{X}^{[i]}$. Using the hourly averaged and 10-minute averaged data sets, the random forest approach was used to train a model for predicting C_n^2 .

In order to improve model effectiveness, the training set was used to determine the location of splits in each of the 250 constituent regression trees. Trees with a variety of minimum nodal observations and total nodal splits were trained. After all trees were trained and incorporated into a final model using the algorithm in Figure 15, the development set was used to determine which minimum number of nodal observation and maximum number of splits lead to the models with the lowest SSE. Once these hyper-parameters were tuned, a final model was trained using the optimal maximum number of regression tree splits and minimum terminal node size. This final model was evaluated using the test set. As a result, the observations used to evaluate final model performance were never used in prior training or tuning. The MATLAB script used to train the hourly averaged random forest model is given in Appendix E, while the 10-minute average training script is given in Appendix F.

4.3.2 Results and Discussion

The tuning process for the random forest model involved determining the most effective maximum number of splits and minimum node size for constituent trees. Based on the literature, the maximum number of splits selected as testing candidates were 2^n for $n = \{0, 1, 2, \dots, 12\}$, such that the maximal number of splits never exceeded the number of observations in each

training set. The minimum node sizes selected for training were 1, 2, 5, 10, 20, 50, and 100 to give a reasonable spread of constituent tree depths. Each combination of these two hyper-parameters was trained using the training set, and evaluated using the development set. First, the hourly averaged data set was used to develop a full random forest model for C_n^2 prediction. The results of this hyper-parameter tuning process are given in Figure 16.

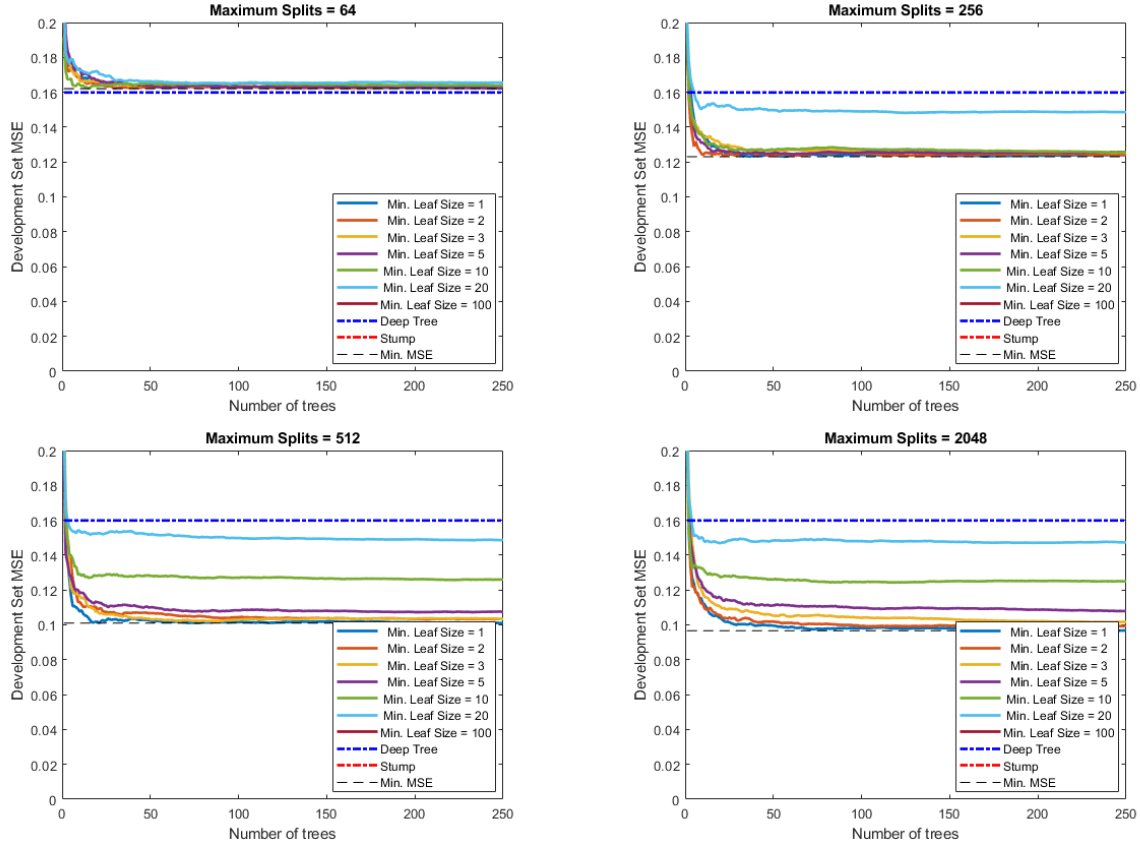


Figure 16: Random Forest Hourly Averaged Data Set Tuning.

The random forest model training results in Figure 16 were examined to determine the optimal number of regression trees to incorporate into the model. Notably, all random forest models exhibited error well below 1, even with a single regression tree component. The number of trees reduced model error for the first 140 trees, but further additions did not materially reduce model error from the 141st tree through the 250th. Even for the first 140 trees, the model did not improve as much as literature suggested it could, indicating that minimum number of observations in each terminal node and number of splits were the dominant hyper parameters for the hourly averaged data set. These results are summarized in Table 6.

Table 6: Model Error of Random Forest Models by Hyper-parameter Combination (Hourly Averaged).

Model Error		Minimum Nodal Observations						
		1	2	5	10	20	50	100
Maximum Number of Splits in Each Tree	1	0.227	0.227	0.227	0.227	0.227	0.227	0.227
	2	0.224	0.224	0.224	0.224	0.224	0.224	0.225
	4	0.216	0.216	0.216	0.216	0.217	0.220	0.221
	8	0.203	0.204	0.205	0.205	0.208	0.211	0.217
	16	0.193	0.194	0.194	0.194	0.197	0.198	0.210
	32	0.179	0.179	0.180	0.180	0.182	0.184	0.206
	64	0.163	0.163	0.162	0.162	0.163	0.167	0.206
	128	0.145	0.144	0.144	0.146	0.143	0.149	0.206
	256	0.124	0.124	0.124	0.126	0.125	0.149	0.205
	512	0.102	0.103	0.104	0.106	0.125	0.148	0.205
	1024	0.099	0.099	0.103	0.107	0.125	0.147	0.206
	2048	0.098	0.100	0.101	0.108	0.126	0.149	0.206
	4096	0.097	0.100	0.101	0.106	0.126	0.150	0.205

Based on the results in Table 6, the optimal hyper-parameters of 1 minimum nodal observation and 4,096 maximum regression tree splits were used to train a final model using both the training and development sets. The number of trees incorporated into the forest was set at 140, as training indicated that error reduction was negligible from further trees. This model was then evaluated using the test set. Model performance was analyzed by comparing the measured C_n^2 observation with the corresponding C_n^2 prediction of the model, given in Figure 17.

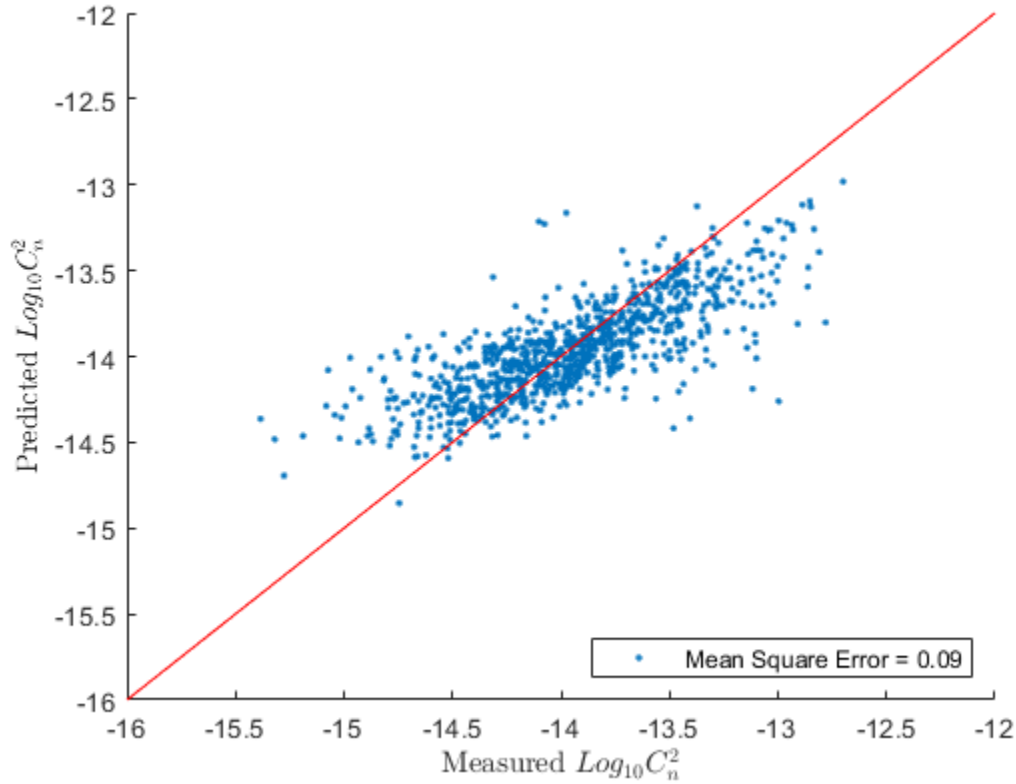


Figure 17: Hourly averaged Random Forest Model Prediction Accuracy.

In order to compare the predicted C_n^2 with the observed C_n^2 , the base 10 logarithm of each was taken in a similar manner to literature [32]. The mean square error in prediction for the $\text{Log}_{10} C_n^2$ was 0.09. Notably, the model prediction was less accurate for lower and higher observed values of C_n^2 , especially those below 5×10^{-15} . Larger model error at lower observed C_n^2 is an issue in current physical models for the near-maritime environment, as in literature [2, 3, 8, 10]. After training a random forest model for the hourly averaged data set, the same tuning and testing approach was performed on the 10-minute averaged data set.

For the 10-minute averaged data set, the same hyper-parameters of maximum number of splits and minimum node size in constituent regression trees were tuned. The same combinations of potential values for these hyper-parameters were trained to develop random forests made up of between 1 and 250 regression trees. The results of this hyper parameter tuning process are given in Figure 18.

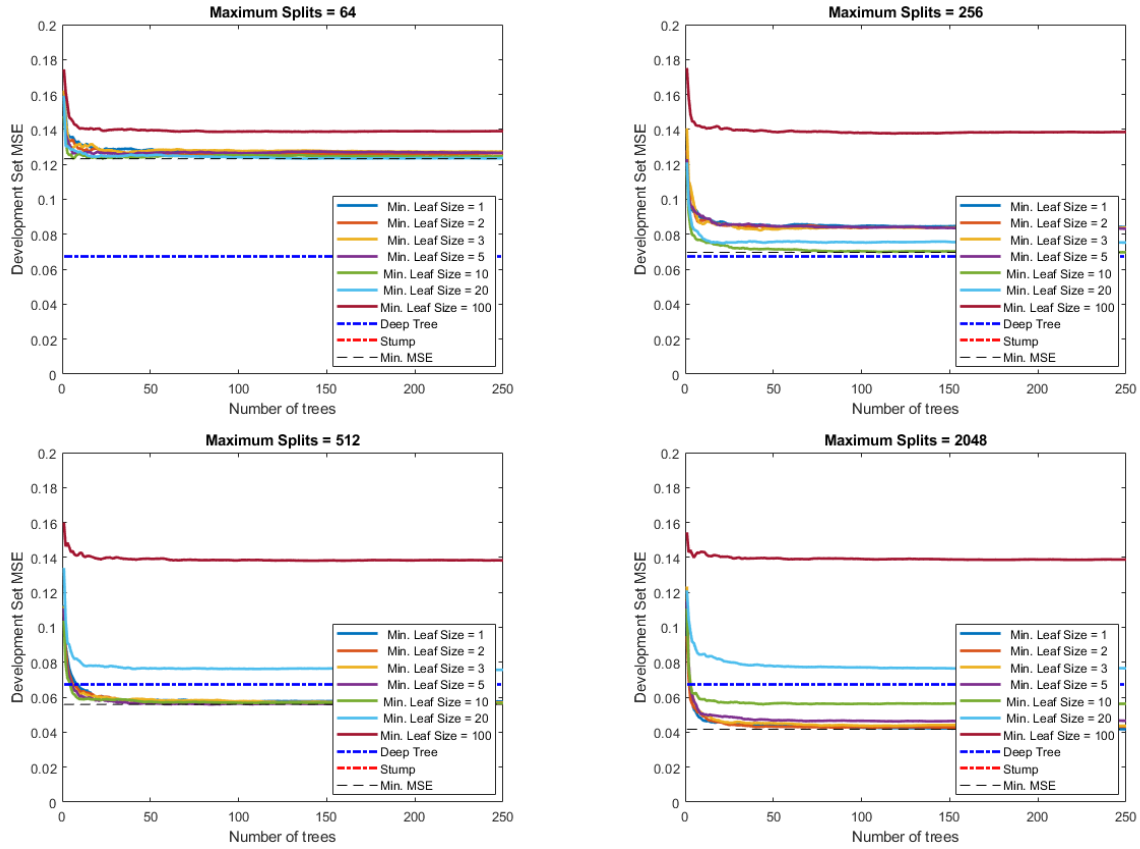


Figure 18: Random Forest 10-minute Averaged Data Set Tuning Results.

The random forest model training in Figure 18 were examined to determine the optimal number of regression trees to incorporate into the model. Notably, models with low numbers of splits in each regression tree failed to outperform the single deep regression tree. The number of splits in each regression tree had more of an impact on reducing model training error than the leaf size. Forests made of regression trees with many splits outperformed both the deep regression tree after around 512 total splits. The number of trees reduced model error for the first 219 trees, but further additions did not reduce model error from the 220th tree through the 250th. These results are summarized in Table 7.

Table 7: Model Error of Random Forest Models by Hyper-parameter Combination (10-minute).

Model Error		Minimum Nodal Observations						
		1	2	5	10	20	50	100
Maximum Number of Splits in Each Tree	1	0.233	0.233	0.232	0.232	0.233	0.233	0.233
	2	0.217	0.217	0.217	0.217	0.216	0.216	0.217
	4	0.208	0.208	0.208	0.208	0.208	0.208	0.208
	8	0.196	0.195	0.196	0.195	0.196	0.196	0.196
	16	0.180	0.179	0.180	0.181	0.178	0.177	0.179
	32	0.157	0.156	0.157	0.156	0.153	0.154	0.151
	64	0.130	0.131	0.129	0.131	0.128	0.121	0.150
	128	0.108	0.107	0.108	0.108	0.096	0.081	0.150
	256	0.079	0.080	0.079	0.078	0.061	0.068	0.149
	512	0.051	0.050	0.050	0.048	0.050	0.068	0.149
	1024	0.037	0.038	0.039	0.040	0.050	0.069	0.150
	2048	0.038	0.038	0.038	0.041	0.050	0.068	0.149
	4096	0.037	0.038	0.039	0.040	0.050	0.068	0.150

Based on the results in Table 7, the optimal hyper-parameters of 1 minimum nodal observation and 1,024 maximum regression tree splits were used to train a final model using both the training and development sets. The observed error in the random forest model trained on 10-minute averaged data was 0.037, indicating that the mean square error in predicting $\log_{10} C_n^2$ was lower than either the boosted or random forest model trained on hourly averaged data. The reduction in model error may be attributed to the highly-variable near maritime environment in which the data was collected. Observed C_n^2 , as well as environmental parameter measurements, changed rapidly throughout the data collection period. As a result, temporal proximity may be important in generating accurate predictions for C_n^2 in the near-maritime environment. Model performance is further analyzed by plotting each measured C_n^2 in the test set against predicted C_n^2 after taking the base-10 logarithm of each in Figure 19.

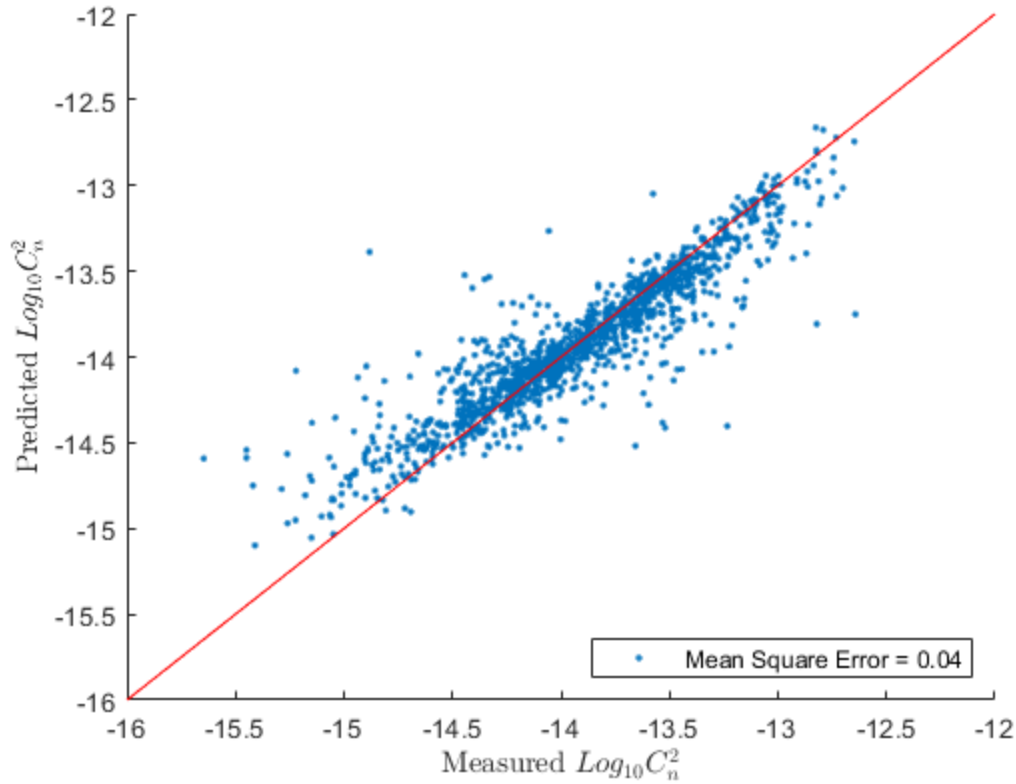


Figure 19: 10-minute Averaged Random Forest Model Prediction Accuracy.

The random forest trained on the 10-minute averaged data set in Figure 19 significantly outperformed the random forest for the hourly averaged data set in Figure 17. The hyper-parameters selected for final model training were similar, both favoring many splits in constituent regression trees with few observations in each terminal node. Performance differences could be attributed to temporal factors in data collection, or to the rapid changes in both optical turbulence and environmental parameters which occur in the near-maritime environment. In order to determine if the added complexity of the random forest model is warranted when compared to the deep single regression tree, their test mean square errors are compared. The random forest mean square error of 0.04 outperformed the deep regression tree, with lower residuals in the middle and upper ranges of observed C_n^2 . Prediction accuracy in the random forest model was less convincing for low observed C_n^2 in the range of 1×10^{-15} . The model over-predicted C_n^2 in this region. Current physics-based models also over-predict C_n^2 in the near-maritime environment, often to a much larger degree [3, 8, 11, 12]. The added prediction accuracy from the inclusion of multiple distinct regression trees helps warrant the added complexity when compared to a deep regression tree. Further regression tree based modeling approaches will be explored through boosted models.

4.4 Boosted Model

In a similar manner to the random forest model discussed in section 4.3, the boosted regression tree model uses a number of distinct regression trees to inform final model predictions. Rather than training a diverse set of regression trees on the training data set, and then taking the average prediction of each regression tree, a boosted model slowly incorporates new trees. A wide variety of boosting approaches exist in literature [13, 14, 27]. The boosting model selected for this study is the least squares boosting algorithm developed by Friedman et al. [27]. This approach makes use of the *gradient* of the SSE objective function in model training. The gradient of a function represents the rate of change in the objective function for the given model and set of observations.

As with the random forest model, two key hyper-parameters are set before training a boosted regression tree. First, the maximum number of splits for regression trees which will be trained and incorporated into the model is set. Additionally, the *learning rate*, a parameter which determines the weight of each new tree added to the boosted model, is set [13]. The learning rate, denoted ν , is between 0 and 1, with $\nu = 1$ incorporating each new tree to maximally reduce SSE [13]. Lower values of ν reduce overall model SSE more slowly to avoid over-fitting to the training data.

4.4.1 Training

The base of the boosted regression tree model is a single regression tree with a fixed number of splits. Each new tree uses the same maximum number of splits as a training hyper-parameter. In order to train new trees and compile them into a final boosted regression tree model, the algorithm in Figure 20 is applied using a pre-set tolerance for boosted model training.

1. *Train a tree T_0 using training data in $\mathbf{X}$. Let the boosted model $F_0 = T_0$.*
2. *Generate predictions $\hat{\mathbf{y}} = F_0(\mathbf{X})$ for observations of $\mathbf{y}$ in the training set.*
3. *Calculate the gradient of model error $\bar{\mathbf{y}}_0 = -\frac{\partial\left(\frac{1}{2}(\mathbf{y}-F_0(\mathbf{X}))^2\right)}{\partial F_0(\mathbf{X})}$.*
4. *For $i = 1, 2, \dots, 250$:*
 - a. *Train a tree $T_i(\mathbf{X})$ to predict $\bar{\mathbf{y}}_{i-1}$, the negative gradient of the error function.*
 - b. *Update the boosted model as $F_i(\mathbf{X}) = F_{i-1}(\mathbf{X}) + \nu(T_i(\mathbf{X}))$.*
 - c. *Calculate the gradient of model error $\bar{\mathbf{y}}_i = -\frac{\partial\left(\frac{1}{2}(\mathbf{y}-F_i(\mathbf{X}))^2\right)}{\partial F_i(\mathbf{X})}$.*
 - d. *If $|\bar{\mathbf{y}}_i|$ falls within the tolerance, let $F(\mathbf{X}) = F_i(\mathbf{X})$ and exit iteration.*
5. *Return the final boosted model $F(\mathbf{X})$.*

Figure 20: Boosted Regression Tree Algorithm.

The algorithm in Figure 20 was applied first to the hourly averaged data set, using many combinations of potential hyper-parameters. As with random forest training, the maximum number of nodal splits in each regression tree was set to 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, and 4096 to capture trees of varying complexity. Additionally, candidate learning rates ν were selected in a similar manner to other literature. The learning rates investigated were 0.01, 0.05, 0.1, 0.2, 0.25, 0.5, and 1. Every potential combination of these two hyper-parameters was considered in model tuning. The MATLAB script used to train the hourly averaged boosted regression tree model is given in Appendix G, while the 10-minute average training script is given in Appendix H.

4.4.2 Results

For each hyper-parameter combination, a boosted model was trained following the algorithm in Figure 20, and the prediction actuary between possible models was compared using the development set. The result of this tuning process is outlined in Figure 21.

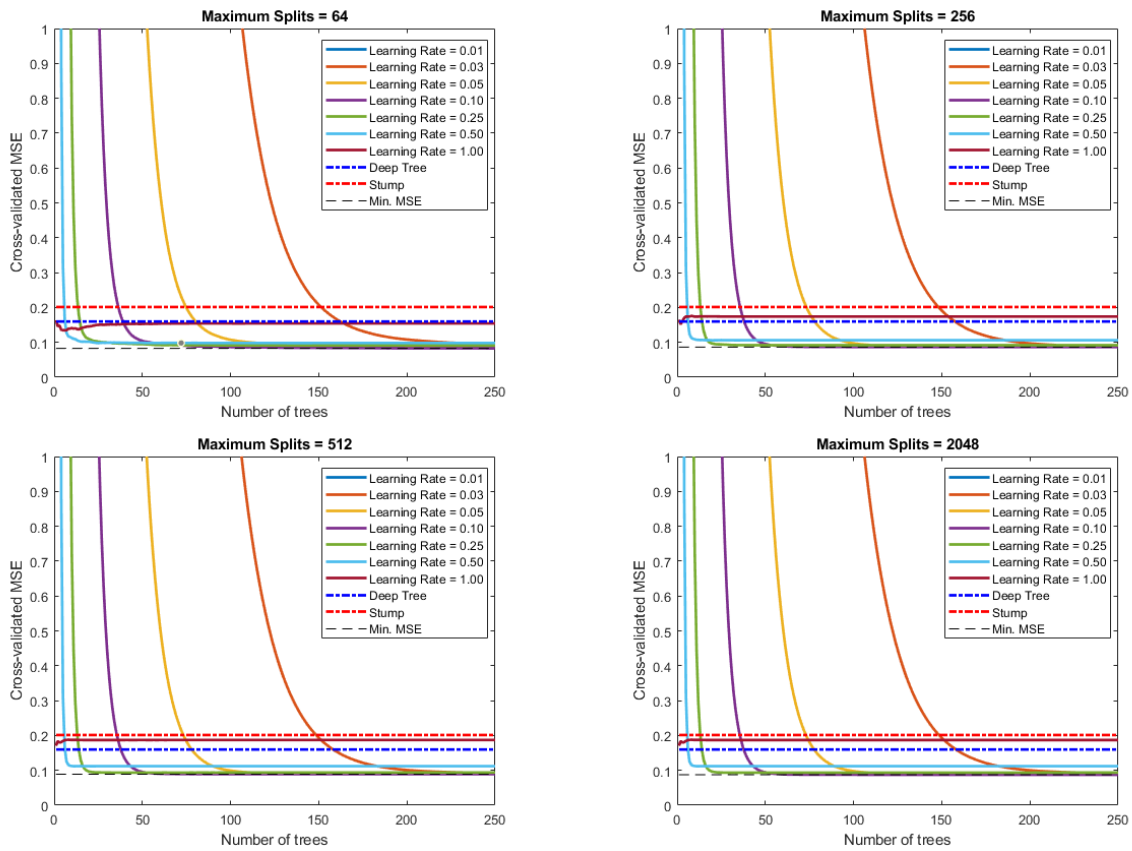


Figure 21: Boosted Regression Tree Hourly Averaged Data Set Tuning.

The boosted regression tree model training in Figure 21 were examined to determine both the optimal hyper-parameter combination and the most appropriate number of regression trees to incorporate into the model. The boosted regression tree models were also each compared against the stump and deep regression trees. With the learning rate set as $\nu = 0.01$, the tuning results

indicated that the deep regression tree outperformed the boosted model. In all other cases, given a sufficient number of included trees, the boosted regression tree significantly outperformed both the deep and stump regression trees. Although the models with higher learning rates lowered their MSE rapidly with few required trees, models with lower learning rates outperformed those with high learning rates after approximately 200 trees were included. The sufficient number of trees differed for each learning rate and maximum number of splits combination. The full hyper-parameter tuning results, which use the development set to evaluate model prediction accuracy, is given in Table 8.

Table 8: Boosted Model Error by Hyper-parameter Combination (Hourly Averaged).

Model Error		Model Learning Rate						
		0.01	0.03	0.05	0.1	0.25	0.5	1
Maximum Number of Splits in Each Tree	1	1.493	0.182	0.174	0.169	0.160	0.154	0.148
	2	1.475	0.170	0.158	0.144	0.128	0.117	0.122
	4	1.461	0.158	0.140	0.125	0.109	0.100	0.124
	8	1.453	0.145	0.125	0.108	0.098	0.103	0.153
	16	1.439	0.129	0.111	0.098	0.090	0.106	0.147
	32	1.425	0.110	0.094	0.087	0.090	0.106	0.159
	64	1.411	0.095	0.084	0.083	0.091	0.098	0.154
	128	1.396	0.086	0.078	0.080	0.084	0.097	0.152
	256	1.381	0.088	0.086	0.087	0.092	0.106	0.174
	512	1.390	0.093	0.091	0.089	0.094	0.112	0.187
	1024	1.388	0.092	0.090	0.087	0.094	0.112	0.187
	2048	1.388	0.092	0.090	0.087	0.094	0.112	0.187
	4096	1.388	0.092	0.090	0.087	0.094	0.112	0.187

Based on the results in Table 8, the optimal hyper-parameter combination of 128 maximum regression tree splits and a learning rate $\nu = 0.05$ was used to train a final model using both the hourly averaged training and development data sets. The number of trees incorporated into the forest was set at 222, as training indicated that error reduction was negligible from further trees. When compared to the random forest trained on the hourly averaged data set, constituent trees for the boosted model had fewer splits, and were thus less complex. The learning rate of $\nu = 0.05$ is in line with expectations suggested by literature [13, 27].

After selecting the hyper-parameter combination which performed best in training, and the corresponding optimal number of included regression trees, a final boosted model was trained on the combined training and development set. This model was then evaluated using the test set, in a similar manner to the prior models in sections 4.1 - 4.3. Model performance was analyzed by comparing the base 10 logarithm of measured C_n^2 observation with the corresponding $\text{Log}_{10} C_n^2$ prediction of the model, given in Figure 22.

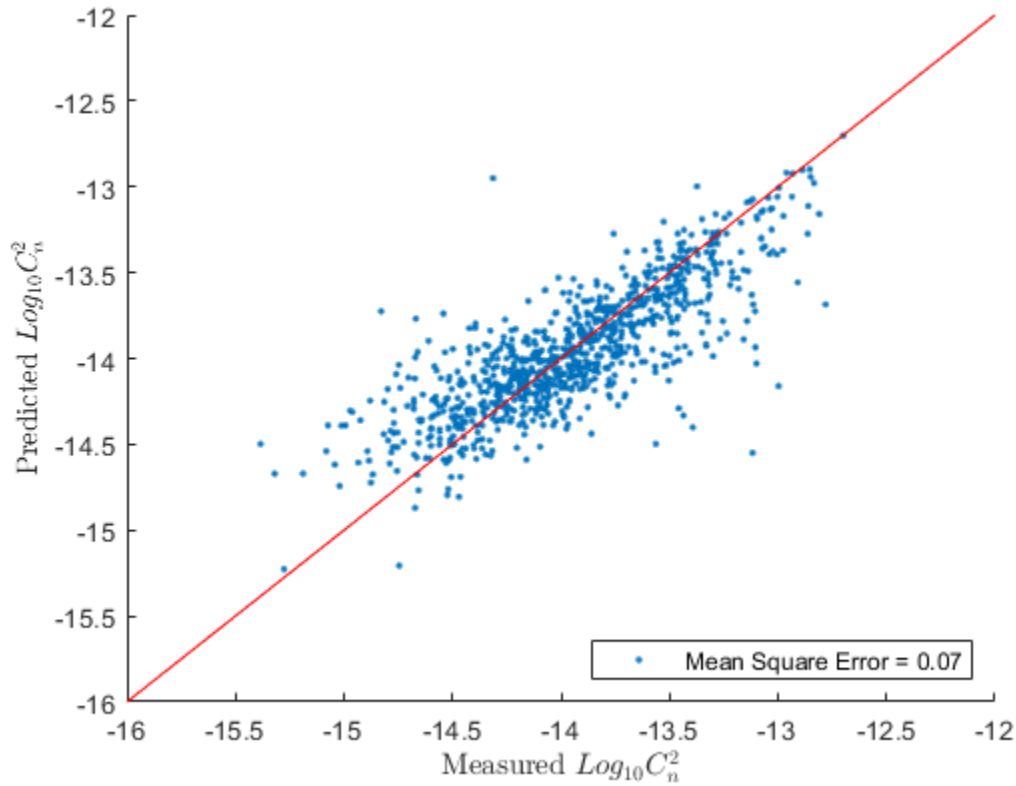


Figure 22: Hourly averaged Boosted Model Prediction Accuracy.

For the boosted regression tree model, the mean square error in prediction for the $\text{Log}_{10} C_n^2$ was 0.07. Both the test MSE and the residuals for extreme observations of C_n^2 were lower for the boosted model than for the corresponding hourly averaged random forest model. Further, the MSE reduction from 0.09 to 0.07 was accomplished using regression trees of lower complexity. The boosted model did make use of a larger number of regression trees when generating a final predicted $\text{Loc}_{10} C_n^2$ than the random forest model. For the boosted model in Figure 22, there were also instances in which C_n^2 was under-predicted by as much as a factor of 10. This could be an issue for model application in the near-maritime environment, as under-prediction of C_n^2 could offer an overly optimistic estimate of interference by local optical turbulence on laser-based systems.

After tuning and training the boosted regression tree model for the hourly averaged data set, a similar tuning and testing approach was performed on the 10-minute averaged data set.

Due to the performance gap between boosted models with a learning rate of 0.05 and those with a learning rate of 0.10, further possible learning rates were added to the set of combinations of hyper-parameters. While the possible maximum number of splits for each regression tree was selected from the same set as in the hourly averaged case, learning rates of 0.07, 0.09, and 0.12 were included for the 10-minute averaged data set model. As in the hourly averaged case, the possible number of trees included in each model ranged from 1 to 250. The development results

of hyper-parameter tuning on the 10-minute averaged training data set are presented in Figure 23.

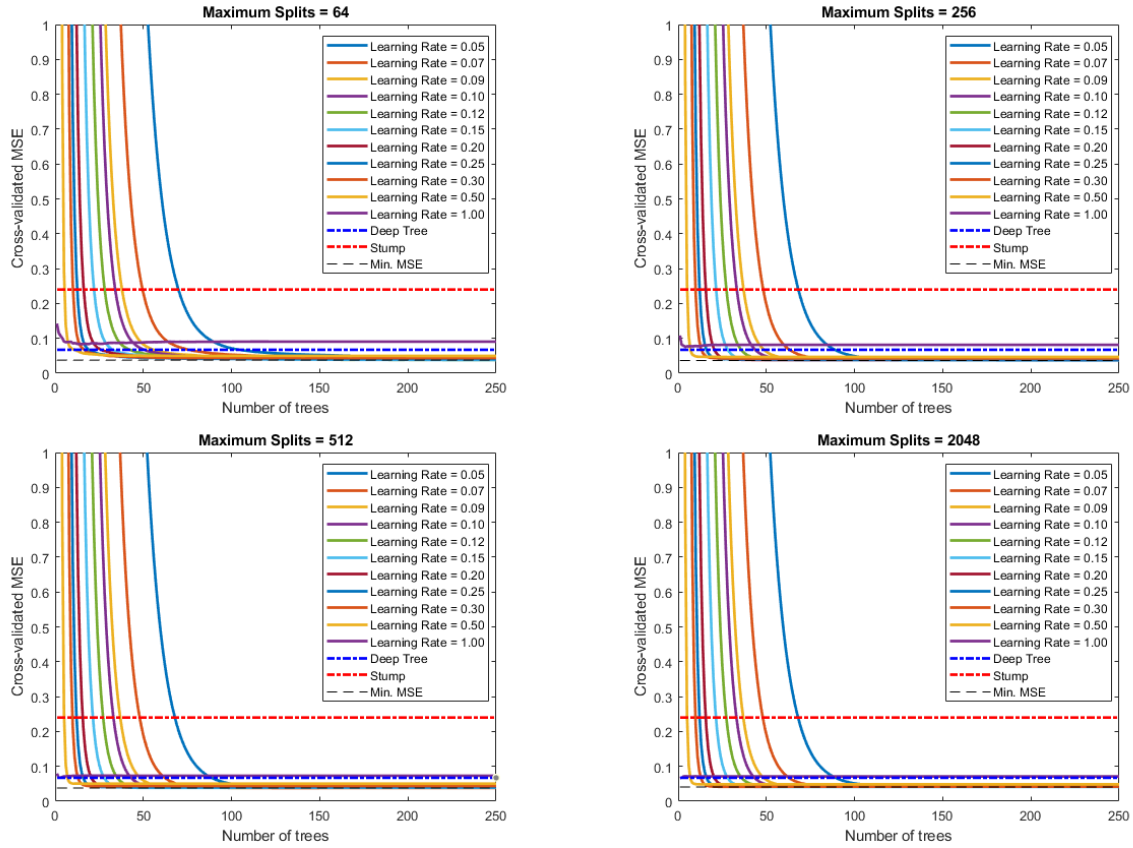


Figure 23: Tuning Results for Boosted Model on 10-minute Averaged Data Set.

The boosted models trained using each combination of potential values for the learning rate and maximum number of regression tree splits exhibited a rapid reduction in model MSE due to the first 20 included trees. There was steep MSE descent when compared to the hourly averaged data, especially for low learning rates. The number of splits in each regression tree had a smaller impact on model MSE than for the hourly averaged case. Although the MSE appears to approach its minimal value for each combination of hyper-parameters quickly, new trees only stopped improving the model after around the 225th tree. A summary of the full development set MSE results for the 10-minute averaged data set are presented in Table 9.

Table 9: Model Error for Booted Models by Hyper-parameter Combination (10-minute).

Model Error		Model Learning Rate										
		0.05	0.07	0.09	0.1	0.12	0.15	0.2	0.25	0.3	0.5	1
Maximum Number of Splits in Each Tree	1	0.176	0.172	0.169	0.168	0.166	0.163	0.159	0.156	0.153	0.147	0.139
	2	0.153	0.141	0.135	0.132	0.128	0.121	0.114	0.106	0.100	0.091	0.088
	4	0.126	0.116	0.110	0.106	0.099	0.093	0.085	0.080	0.074	0.066	0.077
	8	0.104	0.093	0.086	0.081	0.077	0.070	0.064	0.060	0.059	0.058	0.076
	16	0.078	0.067	0.061	0.060	0.055	0.051	0.049	0.049	0.050	0.054	0.084
	32	0.055	0.049	0.047	0.045	0.043	0.043	0.044	0.043	0.047	0.051	0.084
	64	0.044	0.041	0.039	0.039	0.038	0.038	0.041	0.041	0.044	0.050	0.091
	128	0.038	0.037	0.037	0.037	0.038	0.039	0.040	0.039	0.041	0.048	0.082
	256	0.037	0.037	0.038	0.038	0.037	0.037	0.038	0.039	0.042	0.047	0.081
	512	0.040	0.038	0.040	0.039	0.039	0.039	0.043	0.040	0.044	0.050	0.073
	1024	0.042	0.042	0.042	0.041	0.043	0.041	0.043	0.043	0.042	0.048	0.071
	2048	0.042	0.042	0.042	0.041	0.043	0.041	0.043	0.043	0.042	0.048	0.071
4096	0.042	0.042	0.042	0.041	0.043	0.041	0.043	0.043	0.042	0.048	0.071	

Based on the results in Table 9, the optimal hyper-parameter combination of $\nu = 0.09$ and a maximum regression tree split count of 128 was used to train the final boosted model. As before, the training and development sets were merged in order to train the model on the largest possible data set without biasing the test performance. As with the random forest investigated in section 4.3, the use of 10-minute averaged data lead to a substantial reduction in model MSE despite selecting similar hyper-parameters to the hourly averaged model. An MSE reduction from 0.07 to 0.037 from using higher-frequency data represents nearly a 50% reduction in model error. This is further evidence that temporal proximity may be important in generating accurate predictions for C_n^2 in the near-maritime environment. The final boosted model's performance is analyzed by plotting the base 10 logarithm of each measured C_n^2 in the test set against predicted $\text{Log}_{10}C_n^2$ in Figure 24.

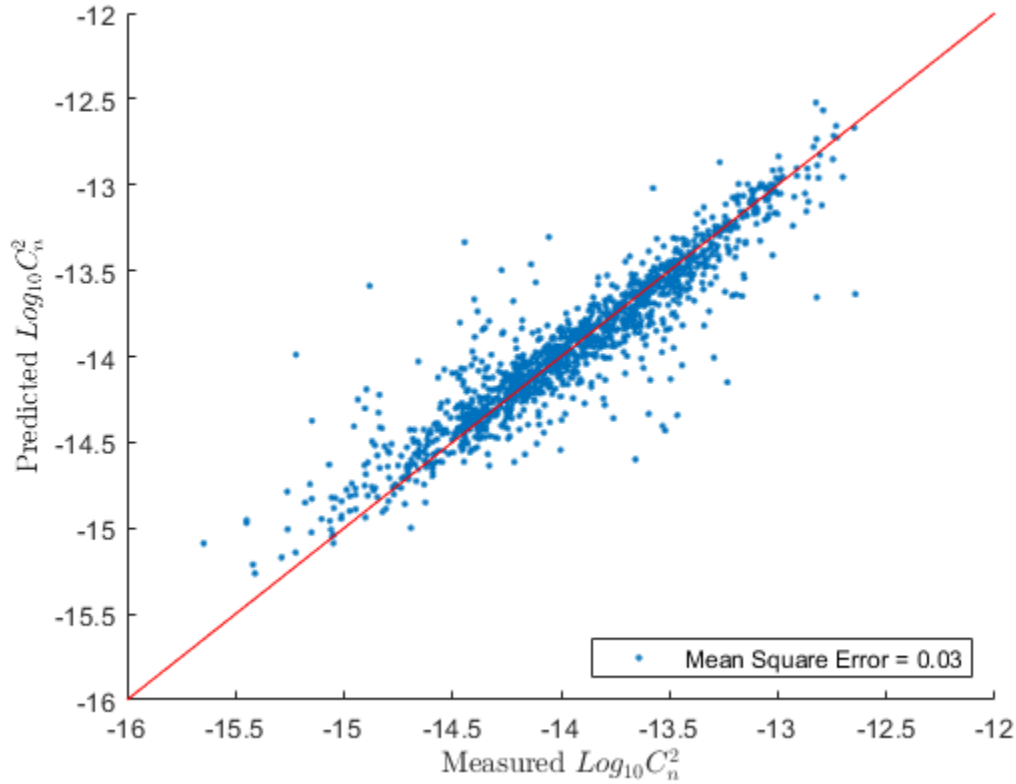


Figure 24: 10-minute Averaged Boosted Model Prediction Accuracy.

The final boosted model trained on the 10-minute averaged data set in Figure 24 outperformed both the random forest for the 10-minute averaged data and the boosted model trained on hourly averaged data. While the random forest had a test MSE of around 0.04, it included trees of greater complexity. The hyper-parameters selected for final model training were similar to the boosted model trained on hourly averaged data. Both favoring lower learning rates and fewer splits in constituent regression trees. While the model is MSE decreased from 0.07 in the hourly averaged case to 0.037, there were still instances in which high observed C_n^2 was under-predicted. These cases of under estimation of optical turbulent effects were more common for high observed C_n^2 than low observed C_n^2 , which differs from results expected by some literature [1, 2, 3, 8, 10, 11, 12].

5 Model Comparison

One key goal of this study was to determine which data-driven model was most appropriate for predicting C_n^2 in the near-maritime environment. The wide range of models and hyper-parameter combinations considered for both the hourly averaged and 10-minute averaged data sets were used to determine the robustness of each approach. After training, tuning, and evaluating the general linear models, regression tree models, random forest models, and boosted models, test error metrics for each model were compared. The standard MSE metric was calculated using the difference between each model's predicted $\text{Log}_{10}C_n^2$ and the corresponding observation of $\text{Log}_{10}C_n^2$ in the test set. In addition, the predicted $\text{Log}_{10}C_n^2$ was plotted against measured $\text{Log}_{10}C_n^2$ to examine the residuals of each model. Model residuals can be investigated to determine if a relationship exists between any environmental parameter and model prediction accuracy. Ideally, residuals would show no clear dependence on measured environmental parameters. If such a trend exists, it may be evidence that a model fails to accurately capture the physics behind near-maritime optical turbulence. Model MSE for both the hourly averaged and 10-minute averaged test data sets is given in Table 10:

Table 10: Model Mean Square Error Comparison for Hourly and 10-minute Averaged Data Sets

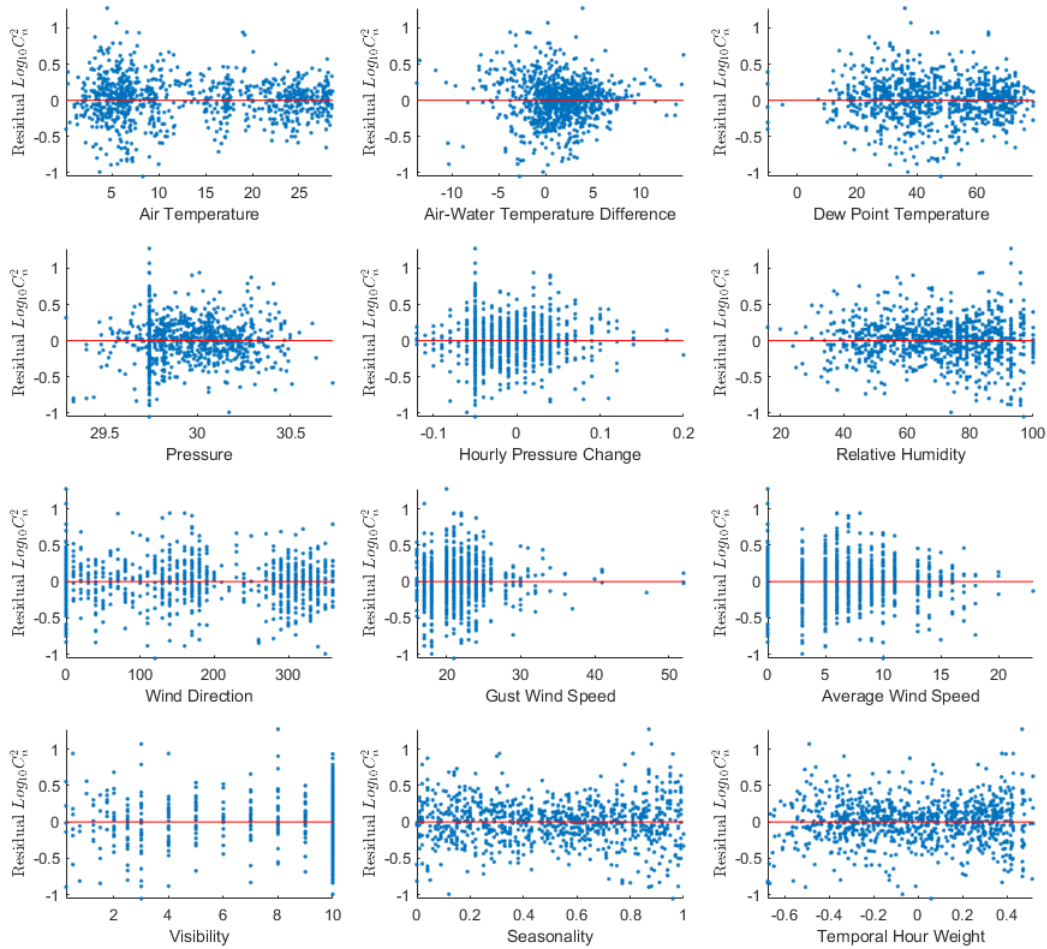
Trained Model		Hourly Averaged Data Set MSE	10-minute Averaged Data Set MSE
General Linear Model	Linear Terms	0.19	0.23
	Interaction Terms	0.17	0.19
Regression Tree	Stump Model	0.26	0.13
	Deep Model	0.13	0.06
	Random Forest	0.089	0.038
	Boosted Model	0.081	0.037

The reported model test error in Table 10 measures each model's ability to accurately predict optical turbulence through $\text{Log}_{10}C_n^2$. Lower mean square error indicates greater model effectiveness. The low reported MSE of the deep regression tree, random forest, and boosted models indicate that non-parametric data-driven approaches may be useful for predicting C_n^2 in the near-maritime environment. All of these models outperformed a tuned general linear model, even when including possible interaction parameters. The robust predictive performance demonstrated by the random forest and boosted models in both the hourly averaged and 10-minute averaged cases is notable. Both show MSE reduction around 35% when compared to a deep regression tree across both data sets.

Another key result reported in Table 10 is the possible significance of temporal effects on C_n^2 prediction. Both the random forest and boosted model demonstrated over 50% error reduction when trained and tested on the 10-minute averaged data set when compared to the hourly averaged data set. Although the boosted model and random forest demonstrate similar error metrics, the trends in their residuals and the relative interpretability differ. These factors can be used to help inform which model is most appropriate for predicting C_n^2 in the near-maritime environment.

5.1 Residuals

To better understand when the random forest and boosted model generate inaccurate predictions for C_n^2 , the residuals of each model are plotted against each measured environmental parameter. This investigation seeks to determine if a clear trend exists between model residuals and any of the measured environmental parameters on which the model is based. The model residuals for the hourly averaged random forest and boosted model are given in Figure 25.



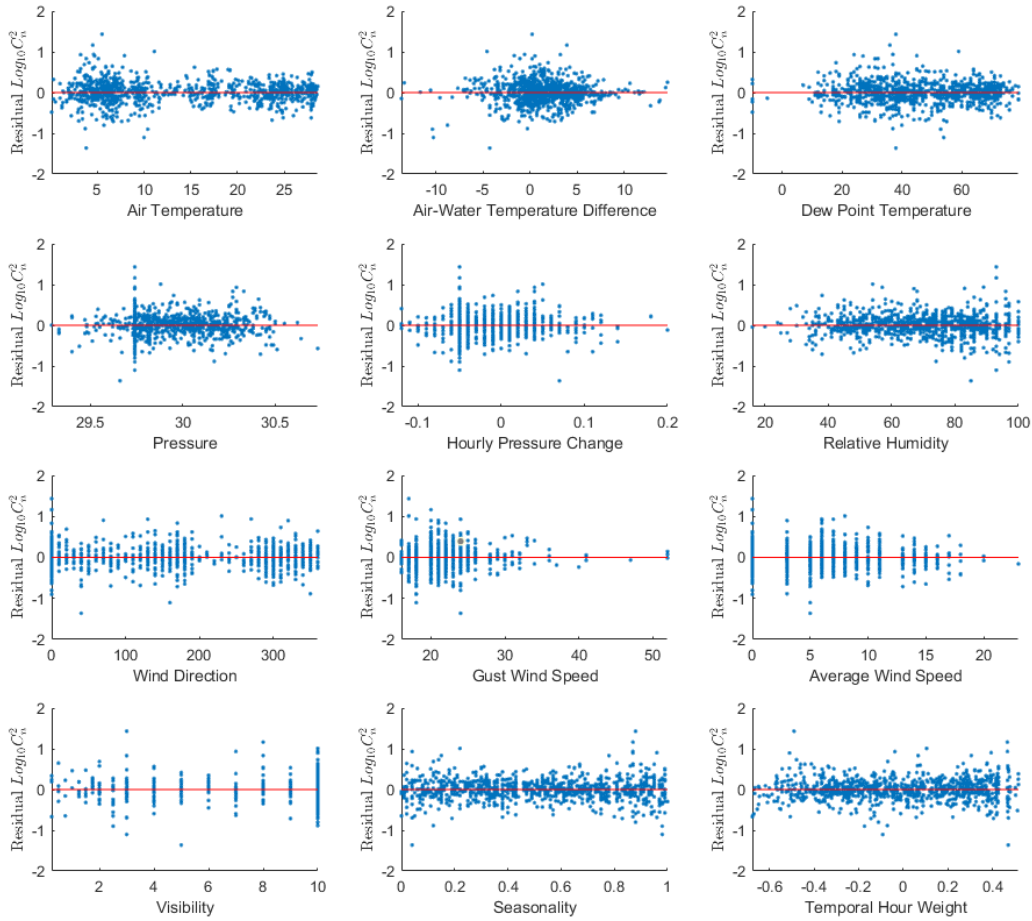
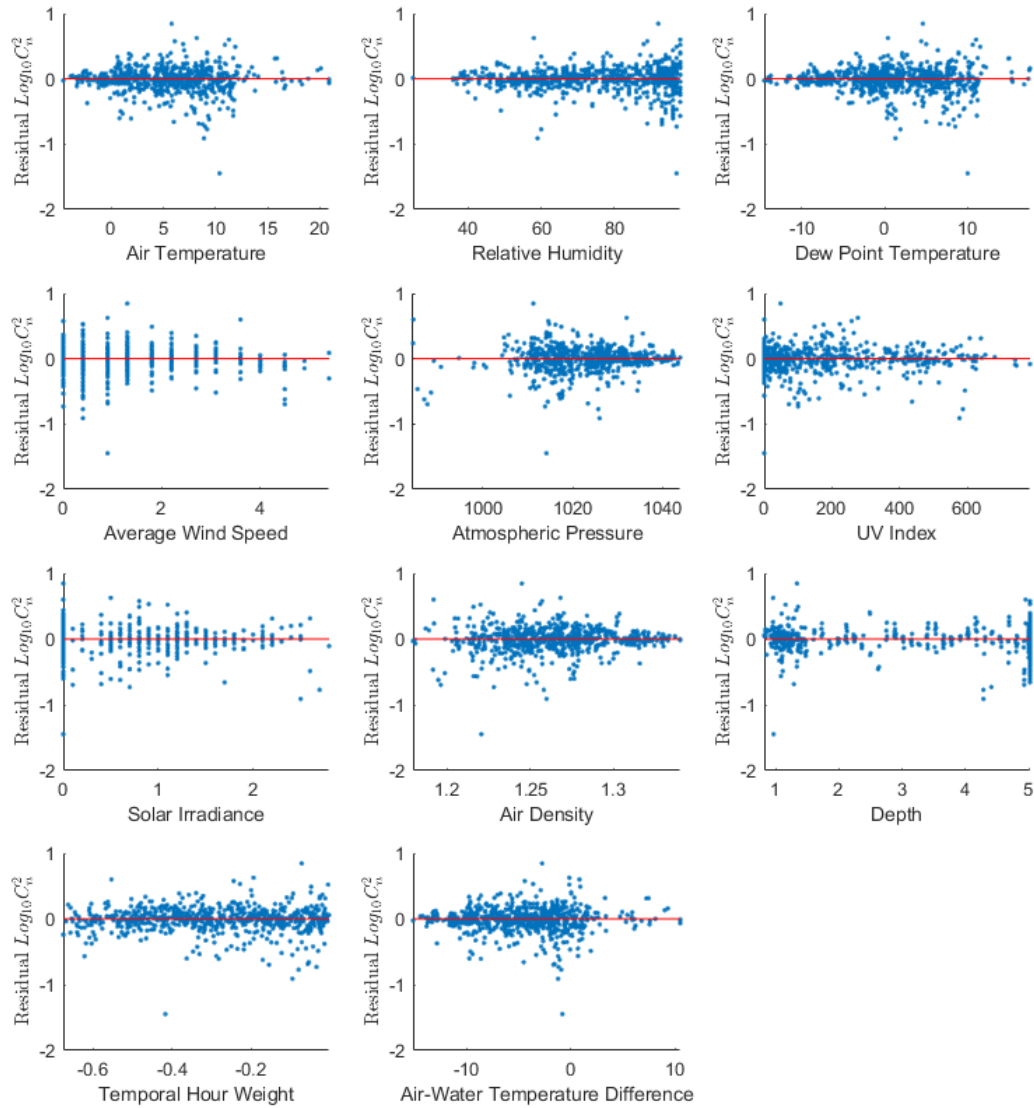


Figure 25: Random Forest (Above) and Boosted Model (Below) Residuals for Hourly Averaged Data Set.

When trained and evaluated using the hourly averaged data set, both the random forest and the boosted model showed higher residuals when the air temperature was close to the water temperature, when relative humidity was above 90%, and when measured air temperature was below 5 [°C]. Current literature suggests that optical turbulence is related to the air-water temperature difference. As both models show some residual dependence on observed air-water temperature difference, the models may fail to fully capture physics-based interactions in the near-maritime environment when the water temperature approaches the air temperature in the late winter and spring. The lack of a clear trend in model residuals for the temporal parameters seasonality and temporal hour weight is also significant. A residual dependence on seasonality would indicate that the models had consistently higher or lower error during a given time of the year. The residuals do appear somewhat lower near a seasonality of 0.5, indicating that the models may be most effective in the early summer. There also appears to be little trend between the average wind speed or wind direction and model residuals.

Model residual dependence is further explored for the random forest and boosted regression tree model using the 10-minute averaged data set. The rainfall data was omitted from this investigation due to instrumentation characteristics which led to non-physical results. The model residuals for the hourly averaged random forest and boosted model are given in Figure 26.



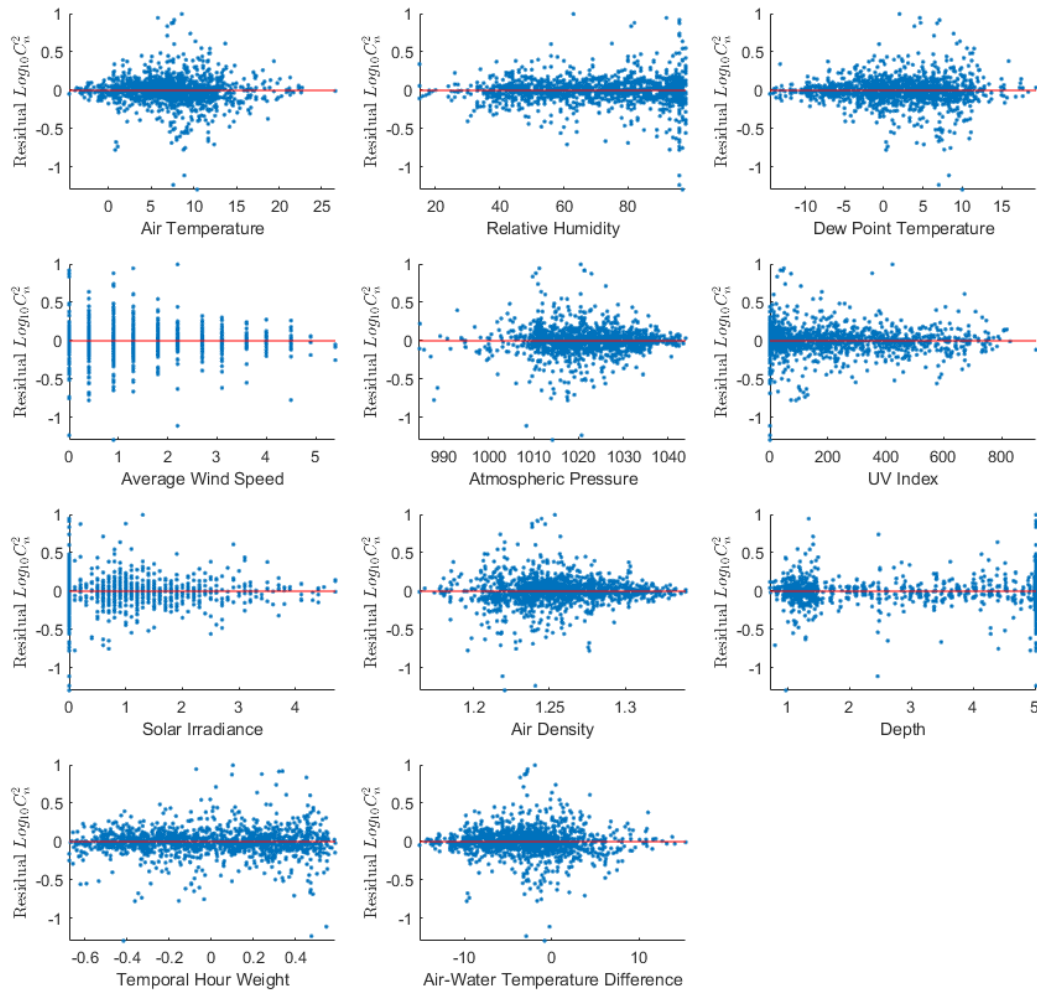


Figure 26: Random Forest (Above) and Boosted Model (Below) Residuals for 10-minute Averaged Data Set.

The 11 environmental parameters in the 10-minute averaged data set shown in Figure 26 were each investigated to determine if a relationship existed between model residuals and parameter measurements. Random forest and boosted model residuals were plotted against each of the remaining environmental parameters in turn. As with the hourly averaged data set, model residuals tended to increase as the air-water temperature difference approached 0. Notably for both the random forest and boosted model, the only observations with residuals above 1 or below -1 occurred within 2.5 [°C] of a 0 [°C] air-water temperature difference. High relative humidity, especially observations above 90%, seemed to correlate with larger model residuals. For both models, the water depth also appeared to influence prediction accuracy. This is in part due to the fact that most observations have an observed depth close to 5 m. Excluding observations near a depth of 5 m, both models demonstrate the largest residuals at depths below 1.5 m.

Both the hourly and 10-minute averaged model residual results reflect a potential relationship between C_n^2 and the air-water temperature difference. This relationship is investigated further in current literature [8, 10, 12]. As the magnitude of residuals for both model's predictions increased above 90% observed relative humidity, the models may fail to capture some of the physics-based interactions which impact local optical turbulence.

The primary goal of this study was to develop a robust machine-learning based model for predicting C_n^2 from environmental parameter measurements in the near-maritime environment. These models had error around 0.09 for the hourly averaged data set and 0.037 for the 10-minute averaged data set. These results offer a significant improvement to the prediction accuracy of some of the bulk-atmospheric models discussed in [8, 11, 12, 32]. As both the random forest and boosted model exhibit similar predictive power on both the hourly and 10-minute averaged data sets, interpretability is considered in selecting a most-appropriate model. Model performance given by Table 10, Figure 25, and Figure 26 indicate that the boosted model may be most appropriate for the prediction of C_n^2 in the near-maritime environment, if interpretability is less of a concern.

5.2 Model Interpretability

In addition to generating accurate predictions for C_n^2 , a robust model for naval applications benefits from interpretability. Understanding how a model reached its prediction can develop the understanding of the physical interactions and relationships which affect optical turbulence in the near-maritime environment. Interpretable models may also have greater compatibility with explicit physics-based models, enabling a combination of data and physics-based assumptions to generate C_n^2 predictions.

The individual regression tree is a highly interpretable model. A user can trace the model prediction from the top level node to any one of the terminal nodes using the nodal splitting criterion. While boosted models use regression trees as components, tracing the process of generating a prediction is much more difficult. Although less intuitive than a single regression tree, boosted models can still be investigated to determine which parameters were most important in reducing model MSE. As random forest models are trained as a simple average of all constituent regression trees, they too can be investigated to identify important parameters. The intuitive construction of random forests allows for easier estimation of the significance of each environmental parameter in producing the final prediction for C_n^2 . The greater interpretability of random forests is a potential advantage for real-world application of data-driven machine learning models for the prediction of C_n^2 . When interpretability is considered, the model performance given by Table 10, Figure 25, and Figure 26 indicate that random forest models may be most appropriate.

6 Relative Feature Importance

In addition to training machine-learning models for the prediction of C_n^2 in the near-maritime environment, this study seeks to investigate which environmental parameters are significant in predicting C_n^2 . Deeper understanding of the physics-based relationships which affect local optical turbulence could be applied in improving the prediction accuracy of current bulk-atmospheric models. Many of these models generate predictions for C_n^2 using local temperature and pressure measurements [5, 6, 8]. Expanding the number of predictors considered when generating predictions for C_n^2 could result in lower model error, especially in the near-maritime environment. Prior studies have indicated the possibility of some relationship between C_n^2 and the air-water temperature difference, temporal hour weight, and local wind speed [2, 5, 6, 8, 10].

Regression tree based models are highly interpretable. Each nodal split is made on a measured environmental parameter. By determining the reduction in model MSE resulting from a given split, a measure of the relative importance of each measured environmental parameter can be obtained. Environmental parameters which contribute to the predictive power of a regression tree based model, including random forest and boosted models, are split on more frequently and offer greater MSE reduction than splits on less important parameters [13, 28, 29, 34].

The secondary objective of this study is to develop new understanding of the potential physical relationships which contribute to optical turbulence. As the most effective models trained in section 4 were the random forest and boosted regression tree trained on the 10-minute averaged data set, these models were selected for further investigation. For both models, the predictor importance of each environmental parameter is computed. The typical MSE function is applied to each node. Next, the reduction in nodal MSE resulting from each nodal split is calculated. The changes in nodal MSE resulting from each nodal split on a selected parameter are added together and divided by the total number of nodal splits which use that parameter.

6.1 Generating Feature Importance Metrics

As both the random forest and boosted models trained in section 4 incorporate regression trees to inform their final predictions, the constituent trees can be investigated to develop relative feature importance metrics. Each model consists of N regression trees $T_1, \dots, T_N$. Within each of these trees, there are $M - 1$ internal nodes, which excludes the top level node of each tree [13, 27]. For each internal node $t \in \{1, \dots, M - 1\}$, the nodal split which minimizes SS_E occurs within one of the 12 environmental parameters $\mathbf{x}_j, j \in \{1, \dots, 12\}$. This split results in two child nodes for the internal node t . The squared relative importance i^2 of environmental parameter $\mathbf{x}_j$ in tree T is the sum of nodal MSE reductions resulting from all splits on $\mathbf{x}_j$ on the $M - 1$ internal nodes, divided by the number of such splits [13].

As random forests use the average of all constituent trees to make a final C_n^2 prediction, the squared relative importance of each environmental parameter can be computed using equation (16).

$$I^2 = \frac{1}{N} \sum_{k=1}^N i_k^2 \quad (16)$$

In equation (16), the squared relative importance of a selected environmental parameter I^2 is determined by averaging the squared relative importance of that parameter within each of the N regression trees in the random forest model. After I^2 is computed for each of the 12 environmental parameters, the squared relative importance values $I_1^2, \dots, I_{12}^2$ for each parameter can be normalized by taking the sum of all I^2 terms and dividing by 12.

For the boosted regression tree model, a similar approach is taken. In the boosted case, the weight of each constituent tree in the model is used to weight the relative importance of a selected environmental parameter i^2 for tree T . The resulting squared relative importance values for the overall model are normalized after weighting the contribution of each tree.

6.2 Results

As the hourly-averaged data set covered a greater time range, from December 2019 until March 2020, the boosted and random forest models trained on this data set were analyzed.

Additionally, this data set contained quality controlled observations of a wider variety of environmental parameters. Further analysis using the 10-minute averaged data set will be more informative once the data set covers a wider range of seasonal conditions. As both the boosted model and random forest model contained regression trees as components, the methodology discussed in Section 6.1 was applied to generate feature importance metrics. Comparisons between the feature importance of each environmental parameter in the data set are presented in Figure 27.

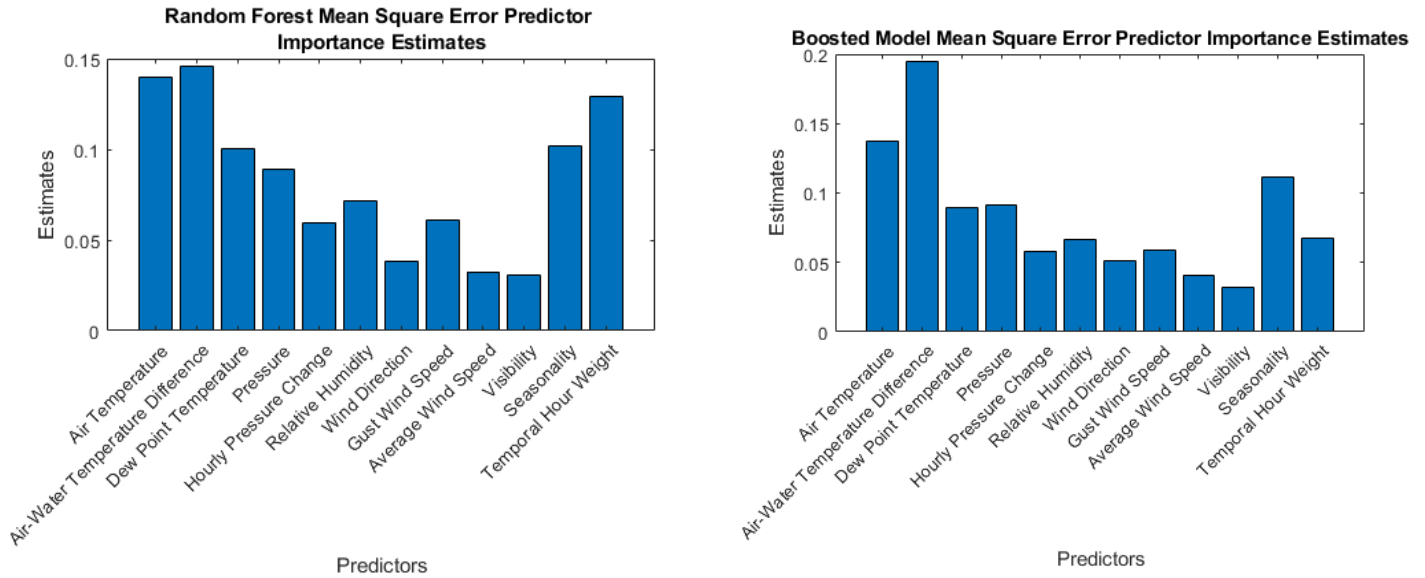


Figure 27: Environmental Parameter Relative Feature Importance for Hourly averaged Random Forest (Left) and Boosted Model (Right).

As expected from similarity theory, absolute atmospheric temperature was an important parameter for predicting C_n^2 [5, 6, 11, 12]. While ensemble predictor importance affirms the significance of this parameter, it also indicates that the air-water temperature difference is equally if not more valuable in generating predictions for C_n^2 . The importance of the air-water

temperature difference was studied in [8, 10, 11]. Additionally, the potential temporal dependence is notable. The random forest and boosted model showed no clear trend between prediction residuals and seasonality or temporal hour weight. The predictor importance analysis performed on both models indicates that these terms are also useful in predicting C_n^2 . This may capture some of the diurnal dependence of C_n^2 in the near-maritime environment discussed in literature [1, 2]. The low relative importance of wind speed is also notable, as some literature has suggested a relationship between wind speed and C_n^2 in the near-maritime environment [10]. The effect of wind speed may be captured through inclusion of one or more other environmental parameters. The results from Figure 27 indicate that, while absolute air temperature is important in predicting C_n^2 , the air-water temperature difference, temporal hour weight, and seasonality also have substantial predictive power.

6.3 Discussion

Comparing the relative efficacy of a wide range of measured environmental parameters in predicting C_n^2 in a low-altitude maritime environment indicates that parameters other than absolute air temperature can be useful in minimizing model error. Based on current similarity and bulk atmospheric theory, air temperature and local pressure are expected to have the greatest predictive power [2, 5, 6]. The analysis in Figure 27 confirms that both of these parameters contribute to a reduction in model MSE, but also suggests that the air-water temperature difference, temporal hour weight, and seasonality could be important parameters to consider when modeling C_n^2 in a low-altitude maritime environment. The analysis further suggests a weak relationship between the measured wind speed and C_n^2 in this domain, however this effect may be captured elsewhere in the models. While relative humidity was an important parameter for minimizing model MSE, the relationship was not as strong as expected for an over-water link [5, 8]. This finding suggests that the maritime environment of the Severn River is distinct from longer over-water links as well as terrestrial links [2, 3, 8, 10, 11, 12]. The relative importance of water temperature, wind speed, and seasonality suggest that a wider range of environmental parameters, including the air-water temperature difference, temporal hour weight, and seasonality, should be considered in models for C_n^2 in the near-maritime environment.

7 Conclusion

The two goals of this study were to train new machine learning models using local environmental measurements to predict optical turbulence through C_n^2 , and to investigate these models to derive new insights into the importance of each of these parameters in predicting C_n^2 . Both goals were accomplished in the near-maritime environment of the Severn River using an 890 m scintillometer link studied over 15 months. After developing two robust data sets, each containing 12 measured environmental parameters and the corresponding measure of optical turbulence C_n^2 , the models trained in Section 4 were tested to determine their predictive power. The results of this analysis inform important conclusions which may be applied to future models for predicting C_n^2 in the near-maritime environment.

All models trained in section 4, including a general linear model, deep regression tree, random forest, and boosted regression tree model, demonstrated some capacity for predicting C_n^2 using environmental parameter measurements. The deep regression tree reduced model test error by over 60% when compared to a stepwise-trained general linear model in both the hourly averaged and 10-minute averaged cases. Further error reductions came from taking ensembles of regression trees in the random forest and boosted models. Both of these models outperformed the single deep regression tree on the hourly averaged and 10-minute averaged data sets. The test errors for both models on the 10-minute averaged data set were nearly identical, despite the boosted model outperforming the random forest on the hourly averaged data set by 8%. A potential temporal dependence was identified in both models, as their error was reduced by over 55% when using 10-minute averaged observations. The hyper-parameters selected for the hourly averaged and 10-minute averaged cases were nearly identical for both models. This could indicate that data used to make C_n^2 predictions in the near-maritime environment should be gathered with higher frequency.

A dramatic error reduction from a test mean square error of 0.19 to 0.037 on the 10-minute averaged data set was accomplished by training ensemble random forest models rather than linear models with interaction terms. This result helps validate the application of new machine learning techniques and models in predicting optical turbulence in the near-maritime environment. As these models were trained on a wide number of environmental parameters, this result also suggests that prediction in the near-maritime environment should incorporate a larger set of environmental parameters. After analyzing the model residuals, it was determined that much of the prediction error occurred when the air-water temperature difference was near 0, and the relative humidity was above 90%. This result should be explored further, to ensure that the machine learning models accurately capture the underlying physics which affects local optical turbulence.

The relative importance of each of these environmental parameters was compared by determining the reduction in model error attributable to each predictor in section 6. These results indicated that the air temperature, air-water temperature difference, seasonality, and temporal hour weight all contribute substantially to model error reduction. Current bulk atmospheric models use local temperature measurements to inform predictions of C_n^2 . Other literature has suggested a potential relationship between C_n^2 and the air-water temperature difference.

Investigating the machine learning models trained in section 4 affirms both of these conclusions and also suggests the importance of temporal hour weight and the time of the year in predicating C_n^2 in the near-maritime environment.

The prediction accuracy of the deep regression tree, random forest, and boosted regression tree models suggest that machine learning can be applied to improve or supplement current models in the context of the near-maritime environment. Additional machine learning models, especially those which make use of time series data, could be applied to further reduce prediction error. The interpretable structure of regression trees and regression tree based models allows for further study into how model predictions are made. This may yield further insights into the physical relationships between optical turbulence and the attributes of the propagation environment.

8 References

1. Tunick, A. D. (1998). The refractive index structure parameter/atmospheric optical turbulence model: CN2 (No. ARL-TR-1615). ARMY RESEARCH LABORATORY ADELPHI MD.
2. Tunick, A., Tikhonov, N., Vorontsov, M., & Carhart, G. (2005). Characterization of optical turbulence (Cn2) data measured at the ARL A_LOT facility (No. ARL-MR-625). US Army Research Laboratory Adelphi United States.
3. Frederickson, P. A., Davidson, K. L., Zeisse, C. R., & Bendall, C. S. (2000). Estimating the refractive index structure parameter over the ocean using bulk methods. *Journal of Applied Meteorology*, 39(10), 1770-1783.
4. Perram, G. P., Cusumano, S. J., Fiorino, S. T., & Hengehold, R. L. (2010). An introduction to laser weapon systems. Directed Energy Professional Society.
5. Ricardo Barrios and Federico Dios (March 7th 2012). Wireless Optical Communications Through the Turbulent Atmosphere: A Review, *Optical Communications Systems*, Narottam Das, IntechOpen, DOI: 10.5772/34740.
6. Andrews, L. C., & Phillips, R. L. (2005). Laser beam propagation through random media. SPIE.
7. Högstöm, U. L. F. (1996). Review of some basic characteristics of the atmospheric surface layer. *Boundary-Layer Meteorology*, 78(3-4), 215-246.
8. Frederickson, P. A., Hammel, S., & Tsintikidis, D. (2006, September). Measurements and modeling of optical turbulence in a maritime environment. In *Atmospheric Optical Modeling, Measurement, and Simulation II* (Vol. 6303, p. 630307). International Society for Optics and Photonics.
9. Foken, T. (2006). 50 years of the Monin–Obukhov similarity theory. *Boundary-Layer Meteorology*, 119(3), 431-447.
10. Mahon, R., Moore, C. I., Burris, H. R., Rabinovich, W. S., Stell, M., & Thomas, L. M. (2009). Analysis of long-term measurements of laser propagation over the Chesapeake Bay. *Applied optics*, 48(12), 2388-2400.
11. Oakley, M., Nelson, C., Brownell, C.J., and Nelson, D., “Experimental analysis of atmospheric optical turbulence and laser beam scintillation in a near-maritime environment.” 13th Directed Energy Systems Symposium, Portsmouth, VA, September 2018.
12. Brownell C., Nelson C., Nelson D., Oakley M., “Experimental Analysis of Atmospheric Optical Turbulence and Laser Beam Scintillation in a Near-Maritime Environment,” Submitted to *Journal of Directed Energy*.

13. Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning: data mining, inference, and prediction. Springer Science & Business Media.
14. James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning* (Vol. 112, pp. 3-7). New York: Springer.
15. Solomatine D., See L., Abrahart R. (2009) Data-Driven Modelling: Concepts, Approaches and Experiences. In: Abrahart R.J., See L.M., Solomatine D.P. (eds) Practical Hydroinformatics. Water Science and Technology Library, Vol 68. Springer, Berlin, Heidelberg
16. Orbanz, P., & Teh, Y. W. (2010). Bayesian Nonparametric Models. Encyclopedia of machine learning, (1).
17. Denil, M., Matheson, D., & De Freitas, N. (2014, January). Narrowing the gap: Random forests in theory and in practice. In International conference on machine learning (pp. 665-673).
18. Google. (n.d.). Map of the United States Naval Academy and Surrounding Areas. Retrieved April 17, 2020.
19. OpenStreetMaps United States Naval Academy. (n.d.). Retrieved April 17, 2020, through Open Data Commons Open Database License (ODbL).
20. United States Naval Academy Oceanography Department EDMAPS Online Data Repository. (n.d.). Retrieved April 17, 2020, from <https://www.usna.edu/Users/oceano/adavies/Research/index.php>
21. Vaisala, O. (2017). Vaisala Weather Transmitter WXT530 Series User Guide. Retrieved April 17, 2020.
22. NOAA National Centers for Environmental Information Station Metadata. (n.d.). Retrieved April 17, 2020, from <https://www.ncdc.noaa.gov/data-access/land-based-station-data/station-metadata>
23. US Department of Commerce, National Oceanic and Atmospheric Administration, National Weather Service, & National Data Buoy Center. (1996, November 8). National Data Buoy Center. Retrieved from <https://www.ndbc.noaa.gov/measdes.shtm>
24. NOAA ESRL Improved Sunrise/Sunset Calculator. (n.d.). Retrieved April 17, 2020, from <https://www.esrl.noaa.gov/gmd/grad/solcalc/sunrise.html>
25. Sun, G., Weng, N. Q., Xiao, L. M., & Ma, C. S. (2005). Profile and character of atmospheric structure constants of refractive index. *Qiangjiguang Yu Lizishu(High Power Laser Part. Beams)*, 17(4), 485-490.
26. Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 1189-1232.

27. Strobl, C., Boulesteix, A. L., Zeileis, A., & Hothorn, T. (2007). Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC bioinformatics*, 8(1), 25.
28. Shalizi, C. (2013). *Advanced data analysis from an elementary point of view*.
29. Dobson, A. J. *An Introduction to Generalized Linear Models*. New York: Chapman & Hall, 1990.
30. McCullagh, P., and J. A. Nelder. *Generalized Linear Models*. New York: Chapman & Hall, 1990.
31. Doss-Hammel, S., Oh, E., Ricklin, J. C., Eaton, F. D., Gilbreath, G. C., & Tsintikidis, D. (2004, October). *A comparison of optical turbulence models*. In *Free-Space Laser Communications IV* (Vol. 5550, pp. 236-246). International Society for Optics and Photonics.
32. Fabien Lauer. *MLweb: A toolkit for machine learning on the web*. Neurocomputing, Elsevier, 2018, 282, pp.74-77.
33. Hofner, B., Hothorn, T., Kneib, T., & Schmid, M. (2011). A framework for unbiased model selection based on boosting. *Journal of Computational and Graphical Statistics*, 20(4), 956-971. Nicodemus, K. K., Malley, J. D., Strobl, C., & Ziegler, A. (2010). The behaviour of random forest permutation-based variable importance measures under predictor correlation. *BMC bioinformatics*, 11(1), 110.

Appendices

Appendix A: Hourly Averaged Data Set MATLAB Script

```
% Chris Jellen
% 21 AUG 19
% Trident Project Data Visualization and Data Set Creation
function [TT_full,Tbl_full,varNames_full,TT,Tbl,varNames] = final_generateDataSet()

data_Scintilometer = importdata('finalData/load-final-Scintilometer-IHR.xlsx'); % load the
scintilometer data from file
timeseries = data_Scintilometer.textdata.Sheet1(1:end,2); %get the time stamps
timeseries = string(timeseries);
timeseries = "0"+timeseries;

Cn2 = data_Scintilometer.data.Sheet1(1:end,1);
isNA = data_Scintilometer.data.Sheet1(1:end,2);

timeseries = replace(timeseries,'010/','10/'); %get into time series format
timeseries = replace(timeseries,'011/','11/'); %get into time series format
timeseries = replace(timeseries,'012/','12/'); %get into time series format
timeseries = replace(timeseries,'/','-'); %get into time series format
timeseries = replace(timeseries,'-1-','-01-'); %get into time series format
timeseries = replace(timeseries,'-2-','-02-'); %get into time series format
timeseries = replace(timeseries,'-3-','-03-'); %get into time series format
timeseries = replace(timeseries,'-4-','-04-'); %get into time series format
timeseries = replace(timeseries,'-5-','-05-'); %get into time series format
timeseries = replace(timeseries,'-6-','-06-'); %get into time series format
timeseries = replace(timeseries,'-7-','-07-'); %get into time series format
timeseries = replace(timeseries,'-8-','-08-'); %get into time series format
timeseries = replace(timeseries,'-9-','-09-'); %get into time series format
timeseries = replace(timeseries,' PM',''); %get into time series format
timeseries = replace(timeseries,' AM',''); %get into time series format
ts = timeseries(~isNA); %create time series

Cn2_clean = rmmissing(Cn2);
[c_idx_missing] = setdiff(Cn2,Cn2_clean); %get the indecies of missing values
ts = timeseries;
ts(c_idx_missing,:)=[];
Tbl = table(ts,Cn2_clean);

[dates_Scintilometer TID] = findgroups(Tbl.ts); %get unique observations for the timeseries
Cn2Backup = Cn2;
Cn2 = splitapply(@mean,Tbl.Cn2_clean,dates_Scintilometer);
Tbl_scintilometer = table(TID,Cn2); %get a final table of averaged 15 minute Cn2 readings and
the corresponding observation time

TT_scin_ts = datetime(TID,'InputFormat',"MM-dd-yyyy HH:mm:ss"); %get the date time vector
TT_scintilometer = table2timetable(Tbl_scintilometer,'RowTimes',TT_scin_ts); %make fine
scintilometer time table

% apply daylight savings time lag to appropriate readings
isDLST = isdst(TT_scintilometer.Time); %get the indecies which must be adjusted
% Daylight Savings Time
TT.Time(isDLST) = (TT_scintilometer.Time(isDLST)+hours(1)); %lag to account for daylight savings
```

```

TT_scintilometer = unique(TT_scintilometer); %remove duplicate readings
Tbl_scintilometer_d1st = timetable2table(TT_scintilometer); %re-convert to a table
TT = TT_scintilometer;

opts = detectImportOptions('finalData/final-WX-GBP-NOAA.xlsx');
opts.SelectedVariableNames = [2]; %get time, humidity, visibility
data_GB = readmatrix('finalData/final-WX-GBP-NOAA.xlsx',opts); %get the time stamps

% adjust the weather station Time series
ts_GB = data_GB(1:end,1); %get the time series
timeseries_GB = ts_GB;
timeseries_GB = replace(timeseries_GB,'T',' '); %get into time series format
timeseries_GB = replace(timeseries_GB,'/','-'); %get into time series format
% timeseries_GB = replace(timeseries_GB,'-1-','-01-'); %get into time series format
% timeseries_GB = replace(timeseries_GB,'-2-','-02-'); %get into time series format
% timeseries_GB = replace(timeseries_GB,'-3-','-03-'); %get into time series format
% timeseries_GB = replace(timeseries_GB,'-4-','-04-'); %get into time series format
% timeseries_GB = replace(timeseries_GB,'-5-','-05-'); %get into time series format
% timeseries_GB = replace(timeseries_GB,'-6-','-06-'); %get into time series format
% timeseries_GB = replace(timeseries_GB,'-7-','-07-'); %get into time series format
% timeseries_GB = replace(timeseries_GB,'-8-','-08-'); %get into time series format
% timeseries_GB = replace(timeseries_GB,'-9-','-09-'); %get into time series format
% timeseries_GB = replace(timeseries_GB,' PM',' '); %get into time series format
% timeseries_GB = replace(timeseries_GB,' AM',' '); %get into time series format
% get the raw data
opts = detectImportOptions('finalData/final-WX-GBP-NOAA.xlsx');
opts.SelectedVariableNames = [43]; %get dew point temperature
opts.DataRange=[1 length(ts_GB)];
dew_Point_Temp = readmatrix('finalData/final-WX-GBP-NOAA.xlsx',opts); %get the time stamps

opts = detectImportOptions('finalData/final-WX-GBP-NOAA.xlsx');
opts.SelectedVariableNames = [44]; %get dry bulb temperature
opts.DataRange=[1 length(ts_GB)];
dry_Bulb_Temp = readmatrix('finalData/final-WX-GBP-NOAA.xlsx',opts); %get the time stamps

opts = detectImportOptions('toLoad_greenburry.xlsx');
opts.SelectedVariableNames = [50]; %get precipitation
opts.DataRange=[1 length(ts_GB)];
pressure = readmatrix('toLoad_greenburry.xlsx',opts); %get the time stamps

opts = detectImportOptions('toLoad_greenburry.xlsx');
opts.SelectedVariableNames = [45]; %get precipitation
opts.DataRange=[1 length(ts_GB)];
rain = readmatrix('toLoad_greenburry.xlsx',opts); %get the time stamps

opts = detectImportOptions('toLoad_greenburry.xlsx');
opts.SelectedVariableNames = [47]; %get pressure change
opts.DataRange=[1 length(ts_GB)];
pressure_Change = readmatrix('toLoad_greenburry.xlsx',opts); %get the time stamps

opts = detectImportOptions('finalData/final-WX-GBP-NOAA.xlsx');
opts.SelectedVariableNames = [49]; %get time, humidity, visibility
opts.DataRange=[1 length(ts_GB)];
relative_Humidity = readmatrix('finalData/final-WX-GBP-NOAA.xlsx',opts); %get the time stamps

```

```

opts = detectImportOptions('finalData/final-WX-GBP-NOAA.xlsx');
opts.SelectedVariableNames = [53]; %get time, humidity, visibility
opts.DataRange=[1 length(ts_GB)];
visibility_nm = readmatrix('finalData/final-WX-GBP-NOAA.xlsx',opts); %get the time stamps

opts = detectImportOptions('finalData/final-WX-GBP-NOAA.xlsx');
opts.SelectedVariableNames = [54]; %get time, humidity, visibility
opts.DataRange=[1 length(ts_GB)];
wet_Bulb_Temperature = readmatrix('finalData/final-WX-GBP-NOAA.xlsx',opts); %get the time stamps

opts = detectImportOptions('finalData/final-WX-GBP-NOAA.xlsx');
opts.SelectedVariableNames = [55]; %get time, humidity, visibility
opts.DataRange=[1 length(ts_GB)];
wind_Direction = readmatrix('finalData/final-WX-GBP-NOAA.xlsx',opts); %get the time stamps

opts = detectImportOptions('finalData/final-WX-GBP-NOAA.xlsx');
opts.SelectedVariableNames = [56]; %get time, humidity, visibility
opts.DataRange=[1 length(ts_GB)];
wind_Speed_Gust = readmatrix('finalData/final-WX-GBP-NOAA.xlsx',opts); %get the time stamps

opts = detectImportOptions('finalData/final-WX-GBP-NOAA.xlsx');
opts.SelectedVariableNames = [57]; %get time, humidity, visibility
opts.DataRange=[1 length(ts_GB)];
wind_Speed_Avg = readmatrix('finalData/final-WX-GBP-NOAA.xlsx',opts); %get the time stamps

TT_GB = timetable(datetime(timeseries_GB,'InputFormat','yyyy-MM-dd
HH:mm:ss'),dew_Point_Temp,dry_Bulb_Temp,pressure,pressure_Change,relative_Humidity,wet_Bulb_Tempe
rature,wind_Direction,wind_Speed_Gust,wind_Speed_Avg,visibility_nm); %get the weather station
time table

data_sunTimes = importdata('finalData/final-solar-local.xlsx'); %load the scintilometer data
from file
% Year data
sunTimes_year = data_sunTimes(1:end,1); %get the time stamps
sunTimes_year = string(sunTimes_year); %convert to string
% Month data
sunTimes_month = data_sunTimes(1:end,2); %get the time stamps
sunTimes_month = string(sunTimes_month); %convert to string
issingleMonth = zeros(size(sunTimes_month));
for i = 1:length(sunTimes_month)
    if strlength(sunTimes_month(i))==1
        issingleMonth(i) = 1; %get single days
    end
end
for i = 1:length(sunTimes_month)
    if issingleMonth(i) == 1
        sunTimes_month(i) = "0"+sunTimes_month(i); %reformat the day
    end
end
% Day data
sunTimes_day = data_sunTimes(1:end,3); %get the time stamps
sunTimes_day = string(sunTimes_day);
issingleDay = zeros(size(sunTimes_day));

```

```

for i = 1:length(sunTimes_day)
    if strlength(sunTimes_day(i))==1
        issingleDay(i) = 1; %get single days
    end
end
for i = 1:length(sunTimes_day)
    if issingleDay(i) == 1
        sunTimes_day(i) = "0"+sunTimes_day(i); %reformat the day
    end
end
% Sunrise Hour Data
sunTimes_rise = data_sunTimes(1:end,4); %get the time stamps
sunTimes_rise = days(sunTimes_rise);
% Sunset Hour Data
sunTimes_set = data_sunTimes(1:end,5); %get the time stamps
sunTimes_set = days(sunTimes_set);
%make year string vector
sunTimes = sunTimes_year+"-"+sunTimes_month+"-"+sunTimes_day;
ts_sunTimes = datetime(sunTimes,'InputFormat','yyyy-MM-dd');
ts_sunTimes_rise = ts_sunTimes+sunTimes_rise';
ts_sunTimes_set = ts_sunTimes+sunTimes_set';
formatOut = 'yyyy-mm-dd HH:MM:SS';
sunTimesRise = string(datestr(ts_sunTimes_rise,formatOut));
sunTimesSet = string(datestr(ts_sunTimes_set,formatOut));
sunTimesSet = repelem(sunTimesSet,24); %repeat elements
sunTimesRise = repelem(sunTimesRise,24); %repeat elements

ts_sunTimes_start = datetime(2019,1,1,0,0,0);
ts_sunTimes_end = datetime(2020,03,31,23,59,0);
ts_sunTimes = ts_sunTimes_start:hours(1):ts_sunTimes_end; %get time series
% make final time table
% sunTimesRise = sunTimesRise';
% sunTimesSet = sunTimesSet';
TT_sunTimes = timetable(datetime(ts_sunTimes,'InputFormat','MM-dd-yyyy
HH:mm:ss'),sunTimesRise,sunTimesSet); %make fine scintilometer time table, where sun times is a
string

NDBC_SN=load('finalData/final-SN-TPLM2-NDBC.txt','%s');
%create the time series
timeseries_NDBC_water = strcat(num2str(NDBC_SN(:,2)),'-',num2str(NDBC_SN(:,3)),'-
',num2str(NDBC_SN(:,1)),{' '},num2str(NDBC_SN(:,4)),':',num2str(NDBC_SN(:,5)),':00');
timeseries_NDBC_water = replace(timeseries_NDBC_water,{'- '},'-'); %get into time series format
timeseries_NDBC_water = replace(timeseries_NDBC_water,'-1-','-01-'); %sget into time series
format
timeseries_NDBC_water = replace(timeseries_NDBC_water,'-2-','-02-'); %get into time series
format
timeseries_NDBC_water = replace(timeseries_NDBC_water,'-3-','-03-'); %get into time series
format
timeseries_NDBC_water = replace(timeseries_NDBC_water,'-4-','-04-'); %get into time series
format
timeseries_NDBC_water = replace(timeseries_NDBC_water,'-5-','-05-'); %get into time series
format
timeseries_NDBC_water = replace(timeseries_NDBC_water,'-6-','-06-'); %get into time series
format

```

```

timeseries_NDBC_water = replace(timeseries_NDBC_water,'-7-','-07-'); %get into time series
format
timeseries_NDBC_water = replace(timeseries_NDBC_water,'-8-','-08-'); %get into time series
format
timeseries_NDBC_water = replace(timeseries_NDBC_water,'-9-','-09-'); %get into time series
format

%read wind direction in degT
wind_Direction = NDBC_SN(:,6); %get the wind direction
toRemove_raw_NDBC_windDir = find(wind_Direction>361); %find missing values
wind_Direction(toRemove_raw_NDBC_windDir) = NaN; %remove missing values

%read wind speed in m/s
wind_Speed = NDBC_SN(:,7); %get the wind speed
toRemove_raw_NDBC_windSpeed = find(wind_Speed>80); %find missing values
wind_Speed(toRemove_raw_NDBC_windSpeed) = NaN; %remove missing values

%read gust wind speed in m/s
raw_NDBC_gustSpeed = NDBC_SN(:,8); %get the air pressure
toRemove_raw_NDBC_gustSpeed = find(raw_NDBC_gustSpeed>60); %find missing values
raw_NDBC_gustSpeed(toRemove_raw_NDBC_gustSpeed) = NaN; %remove missing values

%read pressure in hPa
pressure = NDBC_SN(:,13); %get the air pressure
toRemove_raw_NDBC_pressure = find(pressure>2000); %find missing values
pressure(toRemove_raw_NDBC_pressure) = NaN; %remove missing values

%read air temperature in C
air_Temp = NDBC_SN(:,15); %get the air pressure
toRemove_raw_NDBC_airTemp = find(air_Temp>60); %find missing values
air_Temp(toRemove_raw_NDBC_airTemp) = NaN; %remove missing values

%read water temperature in C
water_Temp = NDBC_SN(:,14); %get the air pressure
toRemove_raw_NDBC_waterTemp = find(water_Temp>60); %find missing values
water_Temp(toRemove_raw_NDBC_waterTemp) = NaN; %remove missing values
air_water_delta = air_Temp-water_Temp; %get air water temperature difference

TT_weatherFull = timetable(datetime(timeseries_NDBC_water,'InputFormat','MM-dd-yyyy
HH:mm:ss'),air_Temp,air_water_delta); %get the weather station time table
TT_weatherFull_new = unique(TT_weatherFull); %remove duplicate times
% TT_weatherFull_new = rmmissing(TT_weatherFull_new); %remove missing time
TT_weatherFull_new = sortrows(TT_weatherFull_new); %sort rows
TT_weatherFull_retime = retime(TT_weatherFull_new,'hourly','nearest'); %retime to the minute

TT_GB_new = unique(TT_GB); %remove duplicate times
% TT_GB_new = rmmissing(TT_GB_new); %remove missing times
TT_GB_new = sortrows(TT_GB_new); %sort rows
TT_GB_retime = retime(TT_GB_new,'regular','nearest','TimeStep',minutes(60)); %retime to the
minute
% TT_GB_retime = retime(TT_GB,'hourly','nearest'); %retime to the minute

TT_full_weather = synchronize(TT_weatherFull_retime,TT_GB_retime,'Intersection'); %retimed
% TT_full_weather = synchronize(TT_scintilometer_new,TT_full_weather);

```

```

TT_full_weather_clean = synchronize(TT_full_weather,TT_sunTimes); %retimed
timeStart = TT_full_weather_clean.Time(1); %get the first time
timeEnd = TT_full_weather_clean.Time(end)-TT_full_weather_clean.Time(1); %get the time across
the sheet
timeEnd.Format = 's'; %make seconds

for i = 1:length(TT_full_weather_clean.Time)
    numVec(i) = TT_full_weather_clean.Time(i)-timeStart; %get the numerator vector in seconds
    Season(i) = round(numVec(i)./timeEnd,2);
end
Season = Season';
TT_full_weather_clean = addvars(TT_full_weather_clean,Season);

for i = 1:length(TT_full_weather_clean.Time)
    timeStart = TT_full_weather_clean.Time(i); %get the first time
    % timeEnd.Format = 's'; %make seconds
    numVec(i) = TT_full_weather_clean.Time(i)-
datetime(TT_full_weather_clean.sunTimesRise(i),'InputFormat','yyyy-MM-dd HH:mm:ss'); %get the
numerator vector in seconds
    temporal_Hour_weight(i) =
numVec(i)./(datetime(TT_full_weather_clean.sunTimesSet(i),'InputFormat','yyyy-MM-dd HH:mm:ss')-
datetime(TT_full_weather_clean.sunTimesRise(i),'InputFormat','yyyy-MM-dd HH:mm:ss'));
end
temporal_Hour_weight = temporal_Hour_weight';
TT_full_weather_clean = addvars(TT_full_weather_clean,temporal_Hour_weight);
TT_full_weather_clean = removevars(TT_full_weather_clean',{'sunTimesRise','sunTimesSet'});
%remove sun times
% TT_full_weather_clean = rmmissing(TT_full_weather_clean); %remove NaNs

TT_full = TT_full_weather_clean; %save the final time table
Cn2_cutoff = (1e-12); %cutoff Cn2 Value
% TT = removevars(TT,'TID');
% remove outliers
S = withtol(TT.Time,minutes(5));
evnParam = TT_full_weather_clean(S,:);
evnParam = evnParam(:,1:end);
S2 = withtol(TT_full_weather_clean.Time,minutes(5));
Cn2Vec = TT(S2,:);
Tbl_full = [Cn2Vec evnParam];
TT_full = Tbl_full;

toRemove = TT_full.Cn2(1:end)>=Cn2_cutoff; %get indexes of outliers
TT_full_clean = TT_full(~toRemove,:); %remove rows with outlier Cn2 values
Tbl_full = timetable2table(TT_full_clean); %make a table
Tbl_full = removevars(Tbl_full,'TID');
varNames_full = {'Time','Cn2','Air Temperature','Air-Water Temperature Difference','Dew Point
Temperature','Dry Bulb Temperature','Pressure','Hourly Pressure Change','Relative Humidity','Wet
Bulb Temperature','Wind Direction','Gust Wind Speed','Average wind
Speed','Visibility','Seasonality','Temporal Hour weight'}; %save the variable names

end

```

Appendix B: 10-minute Averaged Data Set MATLAB Script

```
% Chris Jellen
% 21 AUG 19
% Trident Project Data Visualization and Data Set Creation
function [TT_full,Tbl_full,varNames_full] = final_generateDataSet_WX()

data_Scintilometer = importdata('finalData/load-final-Scintilometer.xlsx'); % load the
scintilometer data from file
timeseries = data_Scintilometer.textdata.Sheet1(1:end,2); %get the time stamps
timeseries = string(timeseries);
timeseries = "0"+timeseries;

Cn2 = data_Scintilometer.data.Sheet1(1:end,1);
isNA = data_Scintilometer.data.Sheet1(1:end,2);

timeseries = replace(timeseries,'010/','10/'); %get into time series format
timeseries = replace(timeseries,'011/','11/'); %get into time series format
timeseries = replace(timeseries,'012/','12/'); %get into time series format
timeseries = replace(timeseries,'/','-'); %get into time series format
timeseries = replace(timeseries,'-1-','-01-'); %get into time series format
timeseries = replace(timeseries,'-2-','-02-'); %get into time series format
timeseries = replace(timeseries,'-3-','-03-'); %get into time series format
timeseries = replace(timeseries,'-4-','-04-'); %get into time series format
timeseries = replace(timeseries,'-5-','-05-'); %get into time series format
timeseries = replace(timeseries,'-6-','-06-'); %get into time series format
timeseries = replace(timeseries,'-7-','-07-'); %get into time series format
timeseries = replace(timeseries,'-8-','-08-'); %get into time series format
timeseries = replace(timeseries,'-9-','-09-'); %get into time series format
timeseries = replace(timeseries,' PM',''); %get into time series format
timeseries = replace(timeseries,' AM',''); %get into time series format
ts = timeseries(~isNA); %create time series

Cn2_clean = rmmissing(Cn2);
[c idx_missing] = setdiff(Cn2,Cn2_clean); %get the indecies of missing values
ts = timeseries;
ts(idx_missing,:)=[];
Tbl = table(ts,Cn2_clean);

[dates_Scintilometer TID] = findgroups(Tbl.ts); %get unique observations for the time series
Cn2Backup = Cn2;
Cn2 = splitapply(@mean,Tbl.Cn2_clean,dates_Scintilometer);
Tbl_scintilometer = table(TID,Cn2); %get a final table of averaged 15 minute Cn2 readings and
the coresponding observation time

TT_scin_ts = datetime(TID,'InputFormat',"MM-dd-yyyy HH:mm:ss"); %get the date time vector
TT_scintilometer = table2timetable(Tbl_scintilometer,'RowTimes',TT_scin_ts); %make fine
scintilometer time table

% apply daylight savings time lag to appropriate readings
isDLST = isdst(TT_scintilometer.Time); %get the indecies which must be adjusted
% Daylight Savings Time
TT.Time(isDLST) = (TT_scintilometer.Time(isDLST)+hours(1)); %lag to account for daylight savings
```

```
TT_scintilometer = unique(TT_scintilometer); %remove duplicate readings
Tbl_scintilometer_dlst = timetable2table(TT_scintilometer); %re-convert to a table
```

```
data_WX = importdata('finalData/load-final-WX-HF-local.xlsx'); %load the WX data from file
timeseries_WX_date = string(data_WX.textdata(3:end,1)); %get the time stamps
timeseries_WX_time = string(datestr(data_WX.data(1:end,1),'HH:MM:SS')); %get the time stamps
% timeseries_WX_isMissing = string(data_WX.textdata(3:end,8)); %get the time stamps
idx_missing = strcmp(data_WX.textdata(3:end,8),'---');
% timeseries_WX_time = "0"+timeseries_WX_time;
timeseries_WX_time = replace(timeseries_WX_time,'010:','10:'); %get into time series format
timeseries_WX_time = replace(timeseries_WX_time,'011:','11:'); %get into time series format
timeseries_WX_time = replace(timeseries_WX_time,'012:','12:'); %get into time series format
spacer = repelem(" ",length(timeseries_WX_date));
timeseries_WX = timeseries_WX_date+spacer+timeseries_WX_time;
% reformat for time series reading
timeseries_WX = replace(timeseries_WX,'/','-'); %get into time series format
timeseries_WX = replace(timeseries_WX,'-1-','-01-'); %get into time series format
timeseries_WX = replace(timeseries_WX,'-2-','-02-'); %get into time series format
timeseries_WX = replace(timeseries_WX,'-3-','-03-'); %get into time series format
timeseries_WX = replace(timeseries_WX,'-4-','-04-'); %get into time series format
timeseries_WX = replace(timeseries_WX,'-5-','-05-'); %get into time series format
timeseries_WX = replace(timeseries_WX,'-6-','-06-'); %get into time series format
timeseries_WX = replace(timeseries_WX,'-7-','-07-'); %get into time series format
timeseries_WX = replace(timeseries_WX,'-8-','-08-'); %get into time series format
timeseries_WX = replace(timeseries_WX,'-9-','-09-'); %get into time series format
timeseries_WX = replace(timeseries_WX,' PM',''); %get into time series format
timeseries_WX = replace(timeseries_WX,' AM',''); %get into time series format
timeseries_WX = replace(timeseries_WX,'/','-'); %get into time series format
% timeseries_WX_time = timeseries_WX_time+":00"; %add time of 0 seconds
ts = timeseries_WX(~idx_missing); %create time series
```

```
opts = detectImportOptions('finalData/load-final-WX-HF-local.xlsx');
opts.SelectedVariableNames = [3]; %get dew point temperature
opts.DataRange=[3 Inf];
air_Temp = (readmatrix('finalData/load-final-WX-HF-local.xlsx',opts)); %get the time stamps

opts = detectImportOptions('finalData/load-final-WX-HF-local.xlsx');
opts.SelectedVariableNames = [6]; %get time, humidity, visibility
opts.DataRange=[3 Inf];
relative_Humidity = (readmatrix('finalData/load-final-WX-HF-local.xlsx',opts)); %get the time stamps

opts = detectImportOptions('finalData/load-final-WX-HF-local.xlsx');
opts.SelectedVariableNames = [7]; %get time, humidity, visibility
opts.DataRange=[3 Inf];
dew_Point_Temperature = (readmatrix('finalData/load-final-WX-HF-local.xlsx',opts)); %get the time stamps

opts = detectImportOptions('finalData/load-final-WX-HF-local.xlsx');
opts.SelectedVariableNames = [8]; %get time, humidity, visibility
opts.DataRange=[3 Inf];
```

```

wind_Speed = (readmatrix('finalData/load-final-WX-HF-local.xlsx',opts)); %get the time stamps

opts = detectImportOptions('finalData/load-final-WX-HF-local.xlsx');
opts.SelectedVariableNames = [17]; %get time, humidity, visibility
opts.DataRange=[3 Inf];
pressure = (readmatrix('finalData/load-final-WX-HF-local.xlsx',opts)); %get the time stamps

opts = detectImportOptions('finalData/load-final-WX-HF-local.xlsx');
opts.SelectedVariableNames = [18]; %get time, humidity, visibility
opts.DataRange=[3 Inf];
rain = (readmatrix('finalData/load-final-WX-HF-local.xlsx',opts)); %get the time stamps

opts = detectImportOptions('finalData/load-final-WX-HF-local.xlsx');
opts.SelectedVariableNames = [20]; %get time, humidity, visibility
opts.DataRange=[3 Inf];
solar_Radiation = (readmatrix('finalData/load-final-WX-HF-local.xlsx',opts)); %get the time stamps

opts = detectImportOptions('finalData/load-final-WX-HF-local.xlsx');
opts.SelectedVariableNames = [23]; %get time, humidity, visibility
opts.DataRange=[3 Inf];
UV_Index = (readmatrix('finalData/load-final-WX-HF-local.xlsx',opts)); %get the time stamps

opts = detectImportOptions('finalData/load-final-WX-HF-local.xlsx');
opts.SelectedVariableNames = [33]; %get time, humidity, visibility
opts.DataRange=[3 Inf];
air_Density = (readmatrix('finalData/load-final-WX-HF-local.xlsx',opts)); %get the time stamps

```

```

air_Temp_clean = rmmissing(air_Temp);
[c idx_missing] = setdiff(air_Temp,air_Temp_clean); %get the indecies of missing values
ts = timeseries_WX;
ts(idx_missing,:)=[];
relative_Humidity(idx_missing,:)=[];
dew_Point_Temperature(idx_missing,:)=[];
wind_Speed(idx_missing,:)=[];
air_Temp(idx_missing,:)=[];
pressure(idx_missing,:)=[];
rain(idx_missing,:)=[];
solar_Radiation(idx_missing,:)=[];
UV_Index(idx_missing,:)=[];
air_Density(idx_missing,:)=[];

Tbl_WX =
table(ts,air_Temp,relative_Humidity,dew_Point_Temperature,wind_Speed,pressure,rain,solar_Radiation,UV_Index,air_Density);

```

```

[dates_WX TID_WX] = findgroups(Tbl_WX.ts); %get unique observations for the time series

```

```

mean_relative_Humidity = splitapply(@mean,Tbl_WX.relative_Humidity,dates_WX);
mean_dew_Point_Temperature = splitapply(@mean,Tbl_WX.dew_Point_Temperature,dates_WX);
mean_wind_Speed = splitapply(@mean,Tbl_WX.wind_Speed,dates_WX);
mean_air_Temp = splitapply(@mean,Tbl_WX.air_Temp,dates_WX);
mean_pressure = splitapply(@mean,Tbl_WX.pressure,dates_WX);

```

```

mean_rain = splitapply(@mean,Tbl_WX.rain,dates_WX);
mean_solar_Radiation = splitapply(@mean,Tbl_WX.solar_Radiation,dates_WX);
mean_UV_Index = splitapply(@mean,Tbl_WX.UV_Index,dates_WX);
mean_air_Density = splitapply(@mean,Tbl_WX.air_Density,dates_WX);

Tbl_WX =
table(TID_WX,mean_air_Temp,mean_relative_Humidity,mean_dew_Point_Temperature,mean_wind_Speed,mean
_pressure,mean_rain,mean_solar_Radiation,mean_UV_Index,mean_air_Density); %get a final table of
averaged 15 minute Cn2 readings and the corresponding observation time

```

```

TT_WX_ts = datetime(TID_WX,'InputFormat','MM-dd-yyyy HH:mm:ss','TimeZone','Etc/UTC'); %get the
date time vector
TT_WX_ts.TimeZone='America/New_York';
TT_WX = table2timetable(Tbl_WX,'RowTimes',TT_WX_ts); %make fine scintilometer time table

Tbl_WX_d1st = timetable2table(TT_WX); %re-convert to a table

```

```

data_SN = importdata('finalData/load-final-SN-EDMAPS.xlsx'); %load the WX data from file

```

```

opts = detectImportOptions('finalData/load-final-SN-EDMAPS.xlsx');
opts.SelectedVariableNames = [1]; %get water temperature in c
opts.DataRange=[5 Inf];
dates_SN = readmatrix('finalData/load-final-SN-EDMAPS.xlsx',opts); %get the time stamps

```

```

opts = detectImportOptions('finalData/load-final-SN-EDMAPS.xlsx');
opts.SelectedVariableNames = [4]; %get water temperature in c
opts.DataRange=[5 Inf];
water_Temp = readmatrix('finalData/load-final-SN-EDMAPS.xlsx',opts); %get the time stamps

```

```

opts = detectImportOptions('finalData/load-final-SN-EDMAPS.xlsx');
opts.SelectedVariableNames = [7]; %get water temperature in c
opts.DataRange=[5 Inf];
depth = readmatrix('finalData/load-final-SN-EDMAPS.xlsx',opts); %get the time stamps

```

```

timeseries_SN = dates_SN; %get the time stamps
timeseries_SN = string(timeseries_SN); %convert to string to get into the correct format for
timetable construction

```

```

% adjust the weather station Time series

```

```

timeseries_SN = replace(timeseries_SN,'/','-'); %get into time series format
timeseries_SN = replace(timeseries_SN,'-1-','-01-'); %get into time series format
timeseries_SN = replace(timeseries_SN,'-2-','-02-'); %get into time series format
timeseries_SN = replace(timeseries_SN,'-3-','-03-'); %get into time series format
timeseries_SN = replace(timeseries_SN,'-4-','-04-'); %get into time series format
timeseries_SN = replace(timeseries_SN,'-5-','-05-'); %get into time series format
timeseries_SN = replace(timeseries_SN,'-6-','-06-'); %get into time series format
timeseries_SN = replace(timeseries_SN,'-7-','-07-'); %get into time series format
timeseries_SN = replace(timeseries_SN,'-8-','-08-'); %get into time series format
timeseries_SN = replace(timeseries_SN,'-9-','-09-'); %get into time series format
timeseries_SN = replace(timeseries_SN,' 1:',' 01:'); %get into time series format
timeseries_SN = replace(timeseries_SN,' 2:',' 02:'); %get into time series format
timeseries_SN = replace(timeseries_SN,' 3:',' 03:'); %get into time series format
timeseries_SN = replace(timeseries_SN,' 4:',' 04:'); %get into time series format
timeseries_SN = replace(timeseries_SN,' 5:',' 05:'); %get into time series format

```

```

timeseries_SN = replace(timeseries_SN,' 6:',' 06:'); %get into time series format
timeseries_SN = replace(timeseries_SN,' 7:',' 07:'); %get into time series format
timeseries_SN = replace(timeseries_SN,' 8:',' 08:'); %get into time series format
timeseries_SN = replace(timeseries_SN,' 9:',' 09:'); %get into time series format
timeseries_SN = replace(timeseries_SN,' PM',''); %get into time series format
timeseries_SN = replace(timeseries_SN,' AM',''); %get into time series format
ts_SN = datetime(timeseries_SN,'InputFormat','dd-MMM-yyyy HH:mm:ss','TimeZone','Etc/UTC');
TT_SN = timetable(ts_SN,water_Temp,depth); %get the water data time table
ts_SN.TimeZone = 'America/New_York';
TT_SN = unique(TT_SN);
TT_SN = sortrows(TT_SN);
TT_SN = retime(TT_SN,'regular','linear','TimeStep',minutes(10)); %get 5 minute interval data,
align with other data sources, use nearest value
% TT_SN = retime(TT_SN,'regular','mean','TimeStep',minutes(10)); %get 5 minute interval data,
align with other data sources, use nearest value
% TT_SN = retime(TT_SN,'regular','nearest','TimeStep',minutes(10)); %get 5 minute interval data,
align with other data sources, use nearest value

Cn2_cutoff = 1e-12;
toRemove = TT_scintilometer.Cn2(1:end)>=Cn2_cutoff; %get indexes of outliers
TT_scintilometer_clean = TT_scintilometer(~toRemove,:); %remove rows with outlier Cn2 Values

data_sunTimes = importdata('finalData/final-solar-local.xlsx'); %load the scintilometer data
from file
% Year data
sunTimes_year = data_sunTimes(1:end,1); %get the time stamps
sunTimes_year = string(sunTimes_year); %convert to string
% Month data
sunTimes_month = data_sunTimes(1:end,2); %get the time stamps
sunTimes_month = string(sunTimes_month); %convert to string
issingleMonth = zeros(size(sunTimes_month));
for i = 1:length(sunTimes_month)
    if strlength(sunTimes_month(i))==1
        issingleMonth(i) = 1; %get single days
    end
end
for i = 1:length(sunTimes_month)
    if issingleMonth(i) == 1
        sunTimes_month(i) = "0"+sunTimes_month(i); %reformat the day
    end
end
% Day data
sunTimes_day = data_sunTimes(1:end,3); %get the time stamps
sunTimes_day = string(sunTimes_day);
issingleDay = zeros(size(sunTimes_day));
for i = 1:length(sunTimes_day)
    if strlength(sunTimes_day(i))==1
        issingleDay(i) = 1; %get single days
    end
end
for i = 1:length(sunTimes_day)
    if issingleDay(i) == 1
        sunTimes_day(i) = "0"+sunTimes_day(i); %reformat the day
    end
end

```

```

end
% Sunrise Hour Data
sunTimes_rise = data_sunTimes(1:end,4); %get the time stamps
sunTimes_rise = days(sunTimes_rise);
% Sunset Hour Data
sunTimes_set = data_sunTimes(1:end,5); %get the time stamps
sunTimes_set = days(sunTimes_set);
%make year string vector
sunTimes = sunTimes_year+"-"+sunTimes_month+"-"+sunTimes_day;
ts_sunTimes = datetime(sunTimes,'InputFormat','yyyy-MM-dd','TimeZone','America/New_York');
ts_sunTimes_rise = ts_sunTimes+sunTimes_rise';
ts_sunTimes_set = ts_sunTimes+sunTimes_set';
formatOut = 'yyyy-mm-dd HH:MM:SS';
sunTimesRise = string(datestr(ts_sunTimes_rise,formatOut));
sunTimesSet = string(datestr(ts_sunTimes_set,formatOut));
sunTimesSet = repelem(sunTimesSet,24*6); %repeat elements
sunTimesRise = repelem(sunTimesRise,24*6); %repeat elements

ts_sunTimes_start = datetime(2019,1,1,0,0,0);
ts_sunTimes_end = datetime(2020,03,31,23,59,0);
ts_sunTimes = ts_sunTimes_start:minutes(10):ts_sunTimes_end; %get time series
% make final time table
% sunTimesRise = sunTimesRise';
% sunTimesSet = sunTimesSet';
TT_sunTimes = timetable(datetime(ts_sunTimes,'InputFormat','MM-dd-yyyy
HH:mm:ss','TimeZone','America/New_York'),sunTimesRise,sunTimesSet); %make fine scintilometer
time table, where sun times is a string

TT_scintilometer.Time.TimeZone = 'America/New_York'; %set the time zone
TT_SN.ts_SN.TimeZone = 'America/New_York'; %set the time zone
TT_sunTimes.Time.TimeZone = 'America/New_York'; %set the time zone
TT_scintilometer_clean = removevars(TT_scintilometer,'TID');
TT_WX = removevars(TT_WX,'TID_WX');

TT_full = synchronize(TT_scintilometer_clean,TT_WX,TT_SN,TT_sunTimes,'Intersection'); %retimed

timeStart = TT_full.Time(1); %get the first time
timeEnd = TT_full.Time(end)-TT_full.Time(1); %get the time across the sheet
timeEnd.Format = 's'; %make seconds

for i = 1:length(TT_full.Time)
    numVec(i) = TT_full.Time(i)-timeStart; %get the numerator vector in seconds
    Season(i) = round(numVec(i)./timeEnd,2);
end
Season = Season';
TT_full = addvars(TT_full,Season);

for i = 1:length(TT_full.Time)
    timeStart = TT_full.Time(i); %get the first time
    % timeEnd.Format = 's'; %make seconds
    numVec(i) = TT_full.Time(i)-datetime(TT_full.sunTimesRise(i),'InputFormat','yyyy-MM-dd
HH:mm:ss','TimeZone','America/New_York'); %get the numerator vector in seconds
    temporal_Hour_weight(i) = numVec(i)./(datetime(TT_full.sunTimesSet(i),'InputFormat','yyyy-MM-
dd HH:mm:ss','TimeZone','America/New_York')-datetime(TT_full.sunTimesRise(i),'InputFormat','yyyy-
MM-dd HH:mm:ss','TimeZone','America/New_York'));

```

```

end
temporal_Hour_weight = temporal_Hour_weight';
TT_full = addvars(TT_full,temporal_Hour_weight);
TT_full = removevars(TT_full,{'sunTimesRise','sunTimesSet'}); %remove sun times
TT_full = rmmissing(TT_full); %remove NaNs

air_water_Temp_Differnece = TT_full.mean_air_Temp-TT_full.water_Temp;
TT_full = addvars(TT_full,air_water_Temp_Differnece);
TT_full = removevars(TT_full,'water_Temp');
Tbl_full = timetable2table(TT_full);

varNames_full = {'Time','Cn2','Air Temperature','Relative Humidity','Dew Point
Temperature','Average Wind Speed','Atmospheric Pressure','Rain','UV Index','Solar
Irradiance','Air Density','Depth','Seasonality','Temporal Hour weight','Air-Water Temperature
Difference'};

end

```

[Published with MATLAB® R2019b](#)

Appendix C: Hourly Averaged General Linear Model Training

```
% Tune GLM

[TT_full,Tbl_full,varNames_full] = final_generateDataSet();
Tbl = removevars(Tbl_full,'Time');
% Tbl = removevars(Tbl,'Season');
Tbl = removevars(Tbl,'dry_Bulb_Temp');
Tbl = removevars(Tbl,'wet_Bulb_Temperature');
varNames_full = varNames_full(3:end);
% varNames_full(ismember(varNames_full,'Seasonality')) = [];
varNames_full(ismember(varNames_full,'Dry Bulb Temperature')) = [];
varNames_full(ismember(varNames_full,'Wet Bulb Temperature')) = [];
varNames_full(ismember(varNames_full,'Dry_Bulb_Temperature')) = [];
varNames_full(ismember(varNames_full,'Wet_Bulb_Temperature')) = [];
Tbl.Cn2=log10(Tbl.Cn2);

rng(1) % For reproducibility
Tbl_save = Tbl; %save full data table
Tbl_test = datasample(Tbl,floor(0.2*size(Tbl,1))); %set aside 20% of overall observations as a
test set
Tbl_trainDev = setdiff(Tbl,Tbl_test); %get the remaining 80% as the training and validation set.
we will train on the training set, tune on the validation set, and finally test on the test set.
n = size(Tbl_trainDev,1); %get the size of the table (number of observations
idx_dev = cvpartition(n,'Holdout',0.25); %generate a development set of 20% of observations,
this is 25% of the training and development set combined.
Tbl_trainDev.Properties.VariableNames =
matlab.lang.makeValidName(Tbl_trainDev.Properties.VariableNames);
X_tbl = removevars(Tbl_trainDev,'Cn2');
X = table2array(X_tbl); %get the predictor variables
Cn2 = Tbl_trainDev.Cn2; %get the response variable
Tbl_trainDev = removevars(Tbl_trainDev,'Cn2');
Tbl_trainDev = addvars(Tbl_trainDev,Cn2);

Md1Final = stepwiseglm(Tbl_trainDev,'linear','upper','linear')

Y_test_idx = Tbl_test.Cn2>-19;
Cn2 = Tbl_test.Cn2(Y_test_idx,:);
Tbl_test = removevars(Tbl_test,'Cn2');
Tbl_test = addvars(Tbl_test,Cn2);
Y_test = Tbl_test.Cn2(Y_test_idx,:);
Y_pred = predict(Md1Final,Tbl_test(Y_test_idx,:));

mseMd1Legend = immse(Y_test,Y_pred);

figure()
hold on
plot(Y_test,Y_pred,'.')
plot([-16 -12],[-16 -12],'r')
xlabel('Measured $Log_{10}$ C_{n}^{2}$','Interpreter','latex')
ylabel('Predicted $Log_{10}$ C_{n}^{2}$','Interpreter','latex')
hL = legend((num2str(mseMd1Legend,'Mean Square Error = %0.2f')),'Location','Southeast');

Md1Final = stepwiseglm(Tbl_trainDev,'linear','upper','interactions')
```

```

Y_test_idx = Tbl_test.Cn2>-19;
Cn2 = Tbl_test.Cn2(Y_test_idx,:);
Tbl_test = removevars(Tbl_test,'Cn2');
Tbl_test = addvars(Tbl_test,Cn2);
Y_test = Tbl_test.Cn2(Y_test_idx,:);
Y_pred = predict(MdlFinal,Tbl_test(Y_test_idx,:));

mseMdlLegend = immse(Y_test,Y_pred);

figure()
hold on
plot(Y_test,Y_pred, '.')
plot([-16 -12],[-16 -12], 'r')
xlabel('Measured $Log_{10}$ C_{n}^{2}$','Interpreter','latex')
ylabel('Predicted $Log_{10}$ C_{n}^{2}$','Interpreter','latex')
hL = legend((num2str(mseMdlLegend,'Mean Square Error = %0.2f')),'Location','Southeast');

```

Published with MATLAB® R2019b

Appendix D: 10-minute Averaged General Linear Model Training

```
% Tune GLM

[TT_full,Tbl_full,varNames_full] = final_generateDataSet_WX();
Tbl = removevars(Tbl_full,'Time');
Tbl = removevars(Tbl,'Season');
% Tbl = removevars(Tbl,'mean_dew_Point_Temperature');
% Tbl = removevars(Tbl,'wet_Bulb_Temperature');

varNames_full = varNames_full(3:end);
varNames_full(ismember(varNames_full,'Seasonality')) = [];
% varNames_full(ismember(varNames_full,'Dew Point Temperature')) = [];
% varNames_full(ismember(varNames_full,'Wet_Bulb_Temperature')) = [];
Tbl.Cn2=log10(Tbl.Cn2);

rng(1) % For reproducibility
Tbl_save = Tbl; %save full data table
Tbl_test = datasample(Tbl,floor(0.2*size(Tbl,1))); %set aside 20% of overall observations as a
test set
Tbl_trainDev = setdiff(Tbl,Tbl_test); %get the remaining 80% as the training and validation set.
We will train on the training set, tune on the validation set, and finally test on the test set.
n = size(Tbl_trainDev,1); %get the size of the table (number of observations
idx_dev = cvpartition(n,'Holdout',0.25); %generate a development set of 20% of observations,
this is 25% of the training and development set combined.
Tbl_trainDev.Properties.VariableNames =
matlab.lang.makeValidName(Tbl_trainDev.Properties.VariableNames);
X_tbl = removevars(Tbl_trainDev,'Cn2');
X = table2array(X_tbl); %get the predictor variables
Cn2 = Tbl_trainDev.Cn2; %get the response variable
Tbl_trainDev = removevars(Tbl_trainDev,'Cn2');
Tbl_trainDev = addvars(Tbl_trainDev,Cn2);

Md1Final = stepwiseglm(Tbl_trainDev,'linear','upper','linear')

Y_test_idx = Tbl_test.Cn2>-19;
Cn2 = Tbl_test.Cn2(Y_test_idx,:);
Tbl_test = removevars(Tbl_test,'Cn2');
Tbl_test = addvars(Tbl_test,Cn2);
Y_test = Tbl_test.Cn2(Y_test_idx,:);
Y_pred = predict(Md1Final,Tbl_test(Y_test_idx,:));

mseMd1Legend = immse(Y_test,Y_pred);

figure()
hold on
plot(Y_test,Y_pred,'.')
plot([-16 -12],[-16 -12],'r')
xlabel('Measured $Log_{10}$ C_{n}^{2}$','Interpreter','latex')
ylabel('Predicted $Log_{10}$ C_{n}^{2}$','Interpreter','latex')
hL = legend((num2str(mseMd1Legend,'Mean Square Error = %0.2f')),'Location','Southeast');

Md1Final = stepwiseglm(Tbl_trainDev,'interactions')
```

```

Y_test_idx = Tbl_test.Cn2>-19;
Cn2 = Tbl_test.Cn2(Y_test_idx,:);
Tbl_test = removevars(Tbl_test,'Cn2');
Tbl_test = addvars(Tbl_test,Cn2);
Y_test = Tbl_test.Cn2(Y_test_idx,:);
Y_pred = predict(MdlFinal,Tbl_test(Y_test_idx,:));

mseMdlLegend = immse(Y_test,Y_pred);

figure()
hold on
plot(Y_test,Y_pred, '.')
plot([-16 -12],[-16 -12], 'r')
xlabel('Measured $Log_{10}$ C_{n}^{2}$','Interpreter','latex')
ylabel('Predicted $Log_{10}$ C_{n}^{2}$','Interpreter','latex')
hL = legend((num2str(mseMdlLegend,'Mean Square Error = %0.2f')),'Location','Southeast');

```

Published with MATLAB® R2019b

Appendix E: Hourly Averaged Random Forest Model Training

```
% Tune Ensemble Learner

[TT_full,Tbl_full,varNames_full] = final_generateDataSet();
Tbl = removevars(Tbl_full,'Time');
% Tbl = removevars(Tbl,'Season');
Tbl = removevars(Tbl,'dry_Bulb_Temp');
Tbl = removevars(Tbl,'wet_Bulb_Temperature');
varNames_full = varNames_full(3:end);
% varNames_full(ismember(varNames_full,'Seasonality')) = [];
varNames_full(ismember(varNames_full,'Dry Bulb Temperature')) = [];
varNames_full(ismember(varNames_full,'Wet Bulb Temperature')) = [];
Tbl.Cn2=log10(Tbl.Cn2);

rng(1) % For reproducibility
Tbl_save = Tbl; %save full data table
Tbl_test = datasample(Tbl,floor(0.2*size(Tbl,1))); %set aside 20% of overall observations as a
test set
Tbl_trainDev = setdiff(Tbl,Tbl_test); %get the remaining 80% as the training and validation set.
we will train on the training set, tune on the validation set, and finally test on the test set.
n = size(Tbl_trainDev,1); %get the size of the table (number of observations
idx_dev = cvpartition(n,'HoldOut',0.25); %generate a development set of 20% of observations,
this is 25% of the training and development set combined.

Md1Deep = fitrtree(Tbl,'Cn2','CrossVal','on','MergeLeaves','off', ...
    'MinParentSize',1,'Surrogate','on');
Md1Stump = fitrtree(Tbl,'Cn2','MaxNumSplits',1,'CrossVal','on', ...
    'Surrogate','on');
n = size(Tbl_trainDev,1);
m = floor(log2(n - 1));
leafSize = [1 2 3 5 10 20 100];
numLR = numel(leafSize);
maxNumSplits = 2.^(0:m);
numMNS = numel(maxNumSplits);
numTrees = 250;
Md1 = cell(numMNS,numLR);

for k = 1:numLR
    for j = 1:numMNS
        t =
templateTree('MaxNumSplits',maxNumSplits(j),'Surrogate','on','MinLeafSize',leafSize(k));
        Md1{j,k} = fitrensemble(Tbl_trainDev,'Cn2','NumLearningCycles',numTrees, ...
            'Learners',t,'CVPartition',idx_dev,'Method','Bag');
    end
end

kf1All = @(x)kfoldLoss(x,'Mode','cumulative');
errorCell = cellfun(kf1All,Md1,'Uniform',false);
error = reshape(cell2mat(errorCell),[numTrees numel(maxNumSplits) numel(leafSize)]);
errorDeep = kfoldLoss(Md1Deep);
errorStump = kfoldLoss(Md1Stump);
```

```

mnsPlot = [5 7 9 10 12];
figure;
for k = 1:5
    figure(k)
    plot(squeeze(error(:,mnsPlot(k),:)), 'LineWidth', 2)
    axis tight
    hold on
    h = gca;
    plot(h.XLim, [errorDeep errorDeep], '-.b', 'LineWidth', 2)
    plot(h.XLim, [errorStump errorStump], '-.r', 'LineWidth', 2)
    plot(h.XLim, min(min(error(:,mnsPlot(k),:))).*[1 1], '--k')
    h.YLim = [0 0.2];
    h.XLim = [0 250];
    xlabel('Number of trees')
    ylabel('Development Set MSE')
    title(sprintf('Maximum Splits = %d', maxNumSplits(mnsPlot(k))))
    hL = legend([cellstr(num2str(leafSize', 'Min. Leaf Size = %d')); ...
        'Deep Tree'; 'Stump'; 'Min. MSE']);
    hL.Position = [0.605357142857143, 0.125793674316198, 0.287499994145972, 0.401190464837211];
    hold off
end

```

```

[minErr, minErrIdxLin] = min(error(:));
[idxNumTrees, idxMNS, idxLR] = ind2sub(size(error), minErrIdxLin);
fprintf('\nMin. MSE = %0.5f', minErr)

fprintf('\nOptimal Parameter Values:\nNum. Trees = %d', idxNumTrees);

fprintf('\nMaxNumSplits = %d\nLearning Rate = %0.2f\n', ...
    maxNumSplits(idxMNS), leafSize(idxLR))

tFinal = templateTree('MaxNumSplits', maxNumSplits(idxMNS), 'Surrogate', 'on');
mdlFinal = fitensemble(Tbl_trainDev, 'Cn2', 'NumLearningCycles', idxNumTrees, ...
    'Learners', tFinal, 'Method', 'Bag');
% saveCompactModel(mdlFinal, 'mdl_Boosted_tuned');

```

```

figure;
dat = (predictorImportance(mdlFinal));
dat = (dat)./(sum(dat));
bar(dat);
title('Mean Square Error Predictor Importance Estimates');
ylabel('Estimates');
xlabel('Predictors');
h = gca;
h.XTickLabel = varNames_full;
h.XTickLabelRotation = 45;
h.TickLabelInterpreter = 'none';
savefig('Boosted MSE Predictor Importance Estimates Local to One')

figure;
dat = (predictorImportance(mdlFinal));
% dat = (dat)./(sum(dat));
bar(dat);
title('Mean Square Error Predictor Importance Estimates');

```

```

ylabel('Estimates');
xlabel('Predictors');
h = gca;
h.XTickLabel = varNames_full;
h.XTickLabelRotation = 45;
h.TickLabelInterpreter = 'none';
savefig('Boosted MSE Predictor Importance Estimates Local')

figure;
dat2 = (Md1Final.oobPermutedPredictorImportance);
% dat2 = (dat2)./(sum(dat2));
bar(dat2);
title('Out-of-bag Permuted Predictor Importance Estimates');
ylabel('Estimates');
xlabel('Predictors');
h = gca;
h.XTickLabel = varNames_full;
h.XTickLabelRotation = 45;
h.TickLabelInterpreter = 'none';
savefig('final Out-of-bag Permuted Predictor Importance Estimates Local')

figure;
dat2 = (Md1Final.oobPermutedPredictorImportance);
dat2 = (dat2)./(sum(dat2));
bar(dat2);
title('Out-of-bag Permuted Predictor Importance Estimates');
ylabel('Estimates');
xlabel('Predictors');
h = gca;
h.XTickLabel = varNames_full;
h.XTickLabelRotation = 45;
h.TickLabelInterpreter = 'none';
savefig('final Out-of-bag Permuted Predictor Importance Estimates Local to One')

%

```

```

Y_test_idx = Tbl_test.Cn2>-19;
Y_pred = predict(Md1Final,Tbl_test(Y_test_idx,:));
Y_test = Tbl_test.Cn2(Y_test_idx,:);
pctErrVec = (abs(10.^(Y_test)-10.^(Y_pred))./10.^(Y_test));
minErrCn2 = 10^(minErr);
pctErr = mean(pctErrVec);
mseMd1Legend = immse(Y_test,Y_pred);

```

```

figure()
hold on
plot(Y_test,Y_pred,'.')
plot([-16 -12],[-16 -12], 'r')
xlabel('Measured  $\log_{10} C_n^{^2}$ ','Interpreter','latex')
ylabel('Predicted  $\log_{10} C_n^{^2}$ ','Interpreter','latex')
hL = legend((num2str(mseMd1Legend,'Mean Square Error = %0.2f')),'Location','Southeast');

residVec = ((Y_test)-(Y_pred));

```

```

% minErrCn2 = 10^-(minErr);
% pctErr = mean(pctErrVec)

Tbl_test_2 = Tbl_test;
Tbl_test_2 = removevars(Tbl_test_2,'Cn2');
% Tbl_test_2 = removevars(Tbl_test_2,'mean_rain');
varNames_full_2 = varNames_full;
% varNames_full_2(6)=[];
figure()
for i = 1:12

    subplot(4,3,i)

    hold on

    toPlot=table2array(Tbl_test_2((Y_test_idx),i));
    plot((toPlot),residVec,'.')
    xlim([min(toPlot) max(toPlot)])
    line(xlim(), [0,0], 'Linewidth', 0.5, 'color', 'r');
    xlabel((varNames_full_2{i}))
        ylabel('Residual $Log_{10} C_{n}^{2}$','Interpreter','latex')

    hold off
end

```

Published with MATLAB® R2019b

Appendix F: 10-minute Averaged Random Forest Model Training

```
% Tune Ensemble Learner

[TT_full,Tbl_full,varNames_full] = final_generateDataSet_WX();
Tbl = removevars(Tbl_full,'Time');
Tbl = removevars(Tbl,'Season');
% Tbl = removevars(Tbl,'mean_dew_Point_Temperature');
% Tbl = removevars(Tbl,'wet_Bulb_Temperature');

varNames_full = varNames_full(3:end);
varNames_full(ismember(varNames_full,'Seasonality')) = [];
% varNames_full(ismember(varNames_full,'Dew Point Temperature')) = [];
% varNames_full(ismember(varNames_full,'Wet Bulb Temperature')) = [];
Tbl.Cn2=log10(Tbl.Cn2);

rng(1) % For reproducibility
Tbl_save = Tbl; %save full data table
Tbl_test = datasample(Tbl,floor(0.2*size(Tbl,1))); %set aside 20% of overall observations as a
test set
Tbl_trainDev = setdiff(Tbl,Tbl_test); %get the remaining 80% as the training and validation set.
We will train on the training set, tune on the validation set, and finally test on the test set.
n = size(Tbl_trainDev,1); %get the size of the table (number of observations
idx_dev = cvpartition(n,'HoldOut',0.25); %generate a development set of 20% of observations,
this is 25% of the training and development set combined.

MdlDeep = fitrtree(Tbl,'Cn2','CrossVal','on','MergeLeaves','off', ...
    'MinParentSize',1,'Surrogate','on');
MdlStump = fitrtree(Tbl,'Cn2','MaxNumSplits',1,'CrossVal','on', ...
    'Surrogate','on');
n = size(Tbl_trainDev,1);
m = floor(log2(n - 1));
leafSize = [1 2 3 5 10 20 100];
numLR = numel(leafSize);
maxNumSplits = 2.^(0:m);
numMNS = numel(maxNumSplits);
numTrees = 250;
Mdl = cell(numMNS,numLR);

for k = 1:numLR
    for j = 1:numMNS
        t =
templateTree('MaxNumSplits',maxNumSplits(j),'Surrogate','on','MinLeafSize',leafSize(k));
        Mdl{j,k} = fitrensemble(Tbl_trainDev,'Cn2','NumLearningCycles',numTrees, ...
            'Learners',t,'CVPartition',idx_dev,'Method','Bag');
    end
end

kf1All = @(x)kfoldLoss(x,'Mode','cumulative');
errorCell = cellfun(kf1All,Mdl,'Uniform',false);
error = reshape(cell2mat(errorCell),[numTrees numel(maxNumSplits) numel(leafSize)]);
```

```
errorDeep = kfoldLoss(MdlDeep);
errorStump = kfoldLoss(MdlStump);
```

```
mnsPlot = [5 7 9 10 12];
figure;
for k = 1:5
    figure(k)
    plot(squeeze(error(:,mnsPlot(k),:)), 'Linewidth', 2)
    axis tight
    hold on
    h = gca;
    plot(h.XLim, [errorDeep errorDeep], '-.b', 'Linewidth', 2)
    plot(h.XLim, [errorStump errorStump], '-.r', 'Linewidth', 2)
    plot(h.XLim, min(min(error(:,mnsPlot(k),:))) .* [1 1], '--k')
    h.YLim = [0 0.2];
    h.XLim = [0 250];
    xlabel('Number of trees')
    ylabel('Development Set MSE')
    title(sprintf('Maximum Splits = %d', maxNumSplits(mnsPlot(k))))
    hL = legend([cellstr(num2str(leafSize, 'Min. Leaf Size = %d')); ...
        'Deep Tree'; 'Stump'; 'Min. MSE']);
    hL.Position = [0.605357142857143, 0.125793674316198, 0.287499994145972, 0.401190464837211];
    hold off
end

[minErr, minErrIdxLin] = min(error(:));
[idxNumTrees, idxMNS, idxLR] = ind2sub(size(error), minErrIdxLin);
fprintf('\nMin. MSE = %0.5f', minErr)

fprintf('\nOptimal Parameter Values:\nNum. Trees = %d', idxNumTrees);

fprintf('\nMaxNumSplits = %d\nLearning Rate = %0.2f\n', ...
    maxNumSplits(idxMNS), leafSize(idxLR))

tFinal = templateTree('MaxNumSplits', maxNumSplits(idxMNS), 'Surrogate', 'on');
MdlFinal = fitensemble(Tbl_trainDev, 'Cn2', 'NumLearningCycles', idxNumTrees, ...
    'Learners', tFinal, 'Method', 'Bag');
% saveCompactModel(MdlFinal, 'mdl_Boosted_tuned');
```

```
figure;
dat = (predictorImportance(MdlFinal));
dat = (dat)./(sum(dat));
bar(dat);
title('Mean Square Error Predictor Importance Estimates');
ylabel('Estimates');
xlabel('Predictors');
h = gca;
h.XTickLabel = varNames_full;
h.XTickLabelRotation = 45;
h.TickLabelInterpreter = 'none';
savefig('Boosted MSE Predictor Importance Estimates Local to One')

figure;
```

```

dat = (predictorImportance(MdlFinal));
% dat = (dat)./(sum(dat));
bar(dat);
title('Mean Square Error Predictor Importance Estimates');
ylabel('Estimates');
xlabel('Predictors');
h = gca;
h.XTickLabel = varNames_full;
h.XTickLabelRotation = 45;
h.TickLabelInterpreter = 'none';
savefig('Boosted MSE Predictor Importance Estimates Local')

figure;
dat2 = (MdlFinal.oobPermutedPredictorImportance);
% dat2 = (dat2)./(sum(dat2));
bar(dat2);
title('Out-of-bag Permuted Predictor Importance Estimates');
ylabel('Estimates');
xlabel('Predictors');
h = gca;
h.XTickLabel = varNames_full;
h.XTickLabelRotation = 45;
h.TickLabelInterpreter = 'none';
savefig('final Out-of-bag Permuted Predictor Importance Estimates Local')

figure;
dat2 = (MdlFinal.oobPermutedPredictorImportance);
dat2 = (dat2)./(sum(dat2));
bar(dat2);
title('Out-of-bag Permuted Predictor Importance Estimates');
ylabel('Estimates');
xlabel('Predictors');
h = gca;
h.XTickLabel = varNames_full;
h.XTickLabelRotation = 45;
h.TickLabelInterpreter = 'none';
savefig('final Out-of-bag Permuted Predictor Importance Estimates Local to One')

```

```

Y_test_idx = Tbl_test.temporal_Hour_weight<0;
Y_pred = predict(MdlFinal,Tbl_test(Y_test_idx,:));
Y_test = Tbl_test.Cn2(Y_test_idx,:);
residVec = ((Y_test)-(Y_pred));
% minErrCn2 = 10^-(minErr);
% pctErr = mean(pctErrVec)

```

```

Tbl_test_2 = Tbl_test;
Tbl_test_2 = removevars(Tbl_test_2,'Cn2');
Tbl_test_2 = removevars(Tbl_test_2,'mean_rain');
varNames_full_2 = varNames_full;
varNames_full_2(6)=[];
figure()
for i = 1:11
    subplot(4,3,i)

```

```

hold on

toPlot=table2array(Tbl_test_2((Y_test_idx),i));
plot((toPlot),residVec,'.')
xlim([min(toPlot) max(toPlot)])
line(xlim(), [0,0], 'Linewidth', 0.5, 'color', 'r');
xlabel((varNames_full_2{i}))
    ylabel('Residual $Log_{10} C_{n}^{2}$','Interpreter','latex')

hold off
end

writetable(array2table(reshape(error(250, :, :), 13, 7)), 'err_RF_HF_local.xlsx')
mseMdlLegend = immse(Y_test, Y_pred);

figure()
hold on
plot(Y_test, Y_pred, '.')
plot([-16 -12], [-16 -12], 'r')
xlabel('Measured $Log_{10} C_{n}^{2}$','Interpreter','latex')
ylabel('Predicted $Log_{10} C_{n}^{2}$','Interpreter','latex')
hL = legend((num2str(mseMdlLegend, 'Mean Square Error = %0.2f')), 'Location', 'Southeast');

```

[Published with MATLAB® R2019b](#)

Appendix G: Hourly Averaged Boosted Model Training

```
% Tune Ensemble Learner

[TT_full,Tbl_full,varNames_full] = final_generateDataSet();
Tbl = removevars(Tbl_full,'Time');
% Tbl = removevars(Tbl,'Season');
Tbl = removevars(Tbl,'dry_Bulb_Temp');
Tbl = removevars(Tbl,'wet_Bulb_Temperature');
varNames_full = varNames_full(3:end);
% varNames_full(ismember(varNames_full,'Seasonality')) = [];
varNames_full(ismember(varNames_full,'Dry Bulb Temperature')) = [];
varNames_full(ismember(varNames_full,'Wet Bulb Temperature')) = [];
Tbl.Cn2=log10(Tbl.Cn2);

rng(1) % For reproducibility
Tbl_save = Tbl; %save full data table
Tbl_test = datasample(Tbl,floor(0.2*size(Tbl,1))); %set aside 20% of overall observations as a
test set
Tbl_trainDev = setdiff(Tbl,Tbl_test); %get the remaining 80% as the training and validation set.
we will train on the training set, tune on the validation set, and finally test on the test set.
n = size(Tbl_trainDev,1); %get the size of the table (number of observations
idx_dev = cvpartition(n,'HoldOut',0.25); %generate a development set of 20% of observations,
this is 25% of the training and development set combined.

MdlDeep = fitrtree(Tbl,'Cn2','CrossVal','on','MergeLeaves','off', ...
    'MinParentSize',1,'Surrogate','on');
MdlStump = fitrtree(Tbl,'Cn2','MaxNumSplits',1,'CrossVal','on', ...
    'Surrogate','on');
n = size(Tbl_trainDev,1);
m = floor(log2(n - 1));
learnRate = [0.01 0.025 0.05 0.1 0.25 0.5 1];
numLR = numel(learnRate);
maxNumSplits = 2.^ (0:m);
numMNS = numel(maxNumSplits);
numTrees = 250;
Mdl = cell(numMNS,numLR);

for k = 1:numLR
    for j = 1:numMNS
        t = templateTree('MaxNumSplits',maxNumSplits(j),'Surrogate','on');
        Mdl{j,k} = fitensemble(Tbl_trainDev,'Cn2','NumLearningCycles',numTrees, ...
            'Learners',t,'CVPartition',idx_dev,'LearnRate',learnRate(k));
    end
end

kflAll = @(x)kfoldLoss(x,'Mode','cumulative');
errorCell = cellfun(kflAll,Mdl,'Uniform',false);
error = reshape(cell2mat(errorCell),[numTrees numel(maxNumSplits) numel(learnRate)]);
errorDeep = kfoldLoss(MdlDeep);
errorStump = kfoldLoss(MdlStump);

writetable(array2table(reshape(error(250,:,:),13,7)), 'err_Boosted_HR.xlsx')
mnsPlot = [5 7 9 10 12];
```

```

figure;
for k = 1:5
    figure(k)
    plot(squeeze(error(:,mnsPlot(k),:)), 'Linewidth',2)
    axis tight
    hold on
    h = gca;
    plot(h.XLim,[errorDeep errorDeep], '-.b', 'Linewidth',2)
    plot(h.XLim,[errorStump errorStump], '-.r', 'Linewidth',2)
    plot(h.XLim,min(min(error(:,mnsPlot(k),:))).*[1 1], '--k')
    h.YLim = [0 1];
    h.XLim = [0 250];
    xlabel('Number of trees')
    ylabel('Development Set MSE')
    title(sprintf('Maximum Splits = %d', maxNumSplits(mnsPlot(k))))
    hL = legend([cellstr(num2str(learnRate','Learning Rate = %0.2f')); ...
        'Deep Tree'; 'Stump'; 'Min. MSE']);
    hL.Position(1) = 0.6;
    hold off
end
% hL = legend([cellstr(num2str(learnRate','Learning Rate = %0.2f')); ...
%           'Deep Tree'; 'Stump'; 'Min. MSE']);
% hL.Position(1) = 0.6;

[minErr,minErrIdxLin] = min(error(:));
[idxNumTrees,idxMNS,idxLR] = ind2sub(size(error),minErrIdxLin);
fprintf('\nMin. MSE = %0.5f',minErr)

fprintf('\nOptimal Parameter Values:\nNum. Trees = %d',idxNumTrees);

fprintf('\nMaxNumSplits = %d\nLearning Rate = %d\n',...
    maxNumSplits(idxMNS),learnRate(idxLR))

tFinal = templateTree('MaxNumSplits',maxNumSplits(idxMNS),'Surrogate','on');
MdlFinal = fitrensemble(Tbl_trainDev,'Cn2','NumLearningCycles',idxNumTrees, ...
    'Learners',tFinal,'LearnRate',learnRate(idxLR));
% saveCompactModel(MdlFinal,'mdl_Boosted_tuned');

% print model
figure;
dat = (predictorImportance(MdlFinal));
dat = (dat)./(sum(dat));
bar(dat);
title('Mean Square Error Predictor Importance Estimates');
ylabel('Estimates');
xlabel('Predictors');
h = gca;
h.XTickLabel = varNames_full;
h.XTickLabelRotation = 45;
h.TickLabelInterpreter = 'none';

figure;
dat = (predictorImportance(MdlFinal));
% dat = (dat)./(sum(dat));

```

```

bar(dat);
title('Mean Square Error Predictor Importance Estimates');
ylabel('Estimates');
xlabel('Predictors');
h = gca;
h.XTickLabel = varNames_full;
h.XTickLabelRotation = 45;
h.TickLabelInterpreter = 'none';

%

Y_test_idx = Tbl_test.Cn2>-19;
Y_pred = predict(MdlFinal,Tbl_test(Y_test_idx,:));
Y_test = Tbl_test.Cn2(Y_test_idx,:);
pctErrVec = (abs(10.^(Y_test)-10.^(Y_pred))./10.^(Y_test));
minErrCn2 = 10^(minErr);
pctErr = mean(pctErrVec)

mseMdlLegend = immse(Y_test,Y_pred);

figure()
hold on
plot(Y_test,Y_pred, '.')
plot([-16 -12],[-16 -12], 'r')
xlabel('Measured $Log_{10} C_{n}^{2}$','Interpreter','latex')
ylabel('Predicted $Log_{10} C_{n}^{2}$','Interpreter','latex')
hL = legend((num2str(mseMdlLegend,'Mean Square Error = %0.2f')),'Location','Southeast');

MdlDeep_2 = fitrtree(Tbl_trainDev,'Cn2','MergeLeaves','off', ...
    'MinParentsSize',2,'Surrogate','on');
Mdlstump_2 = fitrtree(Tbl_trainDev,'Cn2','MaxNumSplits',1, ...
    'Surrogate','on');
Y_test_idx = Tbl_test.Cn2>-19;
Y_pred = predict(MdlDeep_2,Tbl_test(Y_test_idx,:));
Y_test = Tbl_test.Cn2(Y_test_idx,:);
%
mseMdlLegend = immse(Y_test,Y_pred);

figure()
hold on
plot(Y_test,Y_pred, '.')
plot([-16 -12],[-16 -12], 'r')
xlabel('Measured $Log_{10} C_{n}^{2}$','Interpreter','latex')
ylabel('Predicted $Log_{10} C_{n}^{2}$','Interpreter','latex')
hL = legend((num2str(mseMdlLegend,'Mean Square Error = %0.2f')),'Location','Southeast');

% view(MdlDeep)

Y_test_idx = Tbl_test.Cn2>-19;
Y_pred = predict(MdlFinal,Tbl_test(Y_test_idx,:));
Y_test = Tbl_test.Cn2(Y_test_idx,:);
pctErrVec = (abs(10.^(Y_test)-10.^(Y_pred))./10.^(Y_test));
minErrCn2 = 10^(minErr);
pctErr = mean(pctErrVec);
%

```

```

mseMdlLegend = immse(Y_test,Y_pred);

figure()
hold on
plot(Y_test,Y_pred,'.')
plot([-16 -12],[-16 -12],'r')
xlabel('Measured $Log_{10}$ C_{n}^{2}$','Interpreter','latex')
ylabel('Predicted $Log_{10}$ C_{n}^{2}$','Interpreter','latex')
hL = legend((num2str(mseMdlLegend,'Mean Square Error = %0.2f')), 'Location','Southeast');
residVec = ((Y_test)-(Y_pred));
% minErrCn2 = 10^-(minErr);
% pctErr = mean(pctErrVec)

Tbl_test_2 = Tbl_test;
Tbl_test_2 = removevars(Tbl_test_2,'Cn2');
% Tbl_test_2 = removevars(Tbl_test_2,'mean_rain');
varNames_full_2 = varNames_full;
% varNames_full_2(6)=[];
figure()
for i = 1:12

subplot(4,3,i)

hold on

toPlot=table2array(Tbl_test_2((Y_test_idx),i));
plot((toPlot),residVec,'.')
xlim([min(toPlot) max(toPlot)])
line(xlim(), [0,0], 'Linewidth', 0.5, 'Color', 'r');
xlabel((varNames_full_2{i}))
    ylabel('Residual $Log_{10}$ C_{n}^{2}$','Interpreter','latex')

hold off
end

```

Appendix H: 10-minute Averaged Boosted Model Training

```
% Tune Ensemble Learner

[TT_full,Tbl_full,varNames_full] = final_generateDataSet_WX();
Tbl = removevars(Tbl_full,'Time');
Tbl = removevars(Tbl,'Season');
% Tbl = removevars(Tbl,'dry_Bulb_Temp');
% Tbl = removevars(Tbl,'wet_Bulb_Temperature');

varNames_full = varNames_full(3:end);
varNames_full(ismember(varNames_full,'Seasonality')) = [];
% varNames_full(ismember(varNames_full,'Dry_Bulb_Temperature')) = [];
% varNames_full(ismember(varNames_full,'Wet_Bulb_Temperature')) = [];
Tbl.Cn2=log10(Tbl.Cn2);

rng(1) % For reproducibility
Tbl_save = Tbl; %save full data table
Tbl_test = datasample(Tbl,floor(0.2*size(Tbl,1))); %set aside 20% of overall observations as a
test set
Tbl_trainDev = setdiff(Tbl,Tbl_test); %get the remaining 80% as the training and validation set.
We will train on the training set, tune on the validation set, and finally test on the test set.
n = size(Tbl_trainDev,1); %get the size of the table (number of observations
idx_dev = cvpartition(n,'HoldOut',0.25); %generate a development set of 20% of observations,
this is 25% of the training and development set combined.

MdlDeep = fitrtree(Tbl,'Cn2','CrossVal','on','MergeLeaves','off', ...
    'MinParentSize',1,'Surrogate','on');
MdlStump = fitrtree(Tbl,'Cn2','MaxNumSplits',1,'CrossVal','on', ...
    'Surrogate','on');
n = size(Tbl_trainDev,1);
m = floor(log2(n - 1));
learnRate = [0.05 0.07 0.09 0.1 0.12 0.15 0.2 0.25 0.3 0.5 1];
numLR = numel(learnRate);
maxNumSplits = 2.^(0:m);
numMNS = numel(maxNumSplits);
numTrees = 250;
Mdl = cell(numMNS,numLR);

for k = 1:numLR
    for j = 1:numMNS
        t = templateTree('MaxNumSplits',maxNumSplits(j),'Surrogate','on');
        Mdl{j,k} = fitrensemble(Tbl_trainDev,'Cn2','NumLearningCycles',numTrees, ...
            'Learners',t,'CVPartition',idx_dev,'LearnRate',learnRate(k));
    end
end

kf1All = @(x)kfoldLoss(x,'Mode','cumulative');
errorCell = cellfun(kf1All,Mdl,'Uniform',false);
error = reshape(cell2mat(errorCell),[numTrees numel(maxNumSplits) numel(learnRate)]);
errorDeep = kfoldLoss(MdlDeep);
errorStump = kfoldLoss(MdlStump);
```

```

mnsPlot = [5 7 9 10 12 13];
figure;
for k = 1:6
    figure(k)
    plot(squeeze(error(:,mnsPlot(k),:)), 'LineWidth', 2)
    axis tight
    hold on
    h = gca;
    plot(h.XLim, [errorDeep errorDeep], '-.b', 'LineWidth', 2)
    plot(h.XLim, [errorStump errorStump], '-.r', 'LineWidth', 2)
    plot(h.XLim, min(min(error(:,mnsPlot(k),:))).*[1 1], '--k')
    h.YLim = [0 1];
    h.XLim = [0 250];
    xlabel('Number of trees')
    ylabel('Development Set MSE')
    title(sprintf('Maximum Splits = %d', maxNumSplits(mnsPlot(k))))
    hL = legend([cellstr(num2str(learnRate, 'Learning Rate = %0.2f')); ...
        'Deep Tree'; 'Stump'; 'Min. MSE']);
    hL.Position(1) = 0.6;
    hold off
end

[minErr, minErrIdxLin] = min(error(:));
[idxNumTrees, idxMNS, idxLR] = ind2sub(size(error), minErrIdxLin);
fprintf('\nMin. MSE = %0.5f', minErr)

fprintf('\nOptimal Parameter Values:\nNum. Trees = %d', idxNumTrees);

fprintf('\nMaxNumSplits = %d\nLearning Rate = %0.2f\n', ...
    maxNumSplits(idxMNS), learnRate(idxLR))

tFinal = templateTree('MaxNumSplits', maxNumSplits(idxMNS), 'Surrogate', 'on');
MdlFinal = fitensemble(Tbl_trainDev, 'Cn2', 'NumLearningCycles', idxNumTrees, ...
    'Learners', tFinal, 'LearnRate', learnRate(idxLR));
% saveCompactModel(MdlFinal, 'mdl_Boosted_tuned');

figure;
dat = (predictorImportance(MdlFinal));
dat = (dat)./(sum(dat));
bar(dat);
title('Mean Square Error Predictor Importance Estimates');
ylabel('Estimates');
xlabel('Predictors');
h = gca;
h.XTickLabel = varNames_full;
h.XTickLabelRotation = 45;
h.TickLabelInterpreter = 'none';
savefig('Boosted MSE Predictor Importance Estimates Local to One')

figure;
dat = (predictorImportance(MdlFinal));
% dat = (dat)./(sum(dat));

```

```

bar(dat);
title('Mean Square Error Predictor Importance Estimates');
ylabel('Estimates');
xlabel('Predictors');
h = gca;
h.XTickLabel = varNames_full;
h.XTickLabelRotation = 45;
h.TickLabelInterpreter = 'none';
savefig('Boosted MSE Predictor Importance Estimates Local')

Y_test_idx = Tbl_test.Cn2>-19;
Y_pred = predict(MdlFinal,Tbl_test(Y_test_idx,:));
Y_test = Tbl_test.Cn2(Y_test_idx,:);
pctErrVec = (abs(10.^(Y_test)-10.^(Y_pred))./10.^(Y_test));
minErrCn2 = 10^(minErr);
pctErr = mean(pctErrVec)

mseMdlLegend = immse(Y_test,Y_pred);

figure()
hold on
plot(Y_test,Y_pred,'.')
plot([-16 -12],[-16 -12],'r')
xlabel('Measured $Log_{10} C_{n}^{2}$','Interpreter','latex')
ylabel('Predicted $Log_{10} C_{n}^{2}$','Interpreter','latex')
hL = legend((num2str(mseMdlLegend,'Mean Square Error = %0.2f')),'Location','Southeast');

residVec = ((Y_test)-(Y_pred));
% minErrCn2 = 10^(minErr);
% pctErr = mean(pctErrVec)

residVec = ((Y_test)-(Y_pred));
% minErrCn2 = 10^(minErr);
% pctErr = mean(pctErrVec)

Tbl_test_2 = Tbl_test;
Tbl_test_2 = removevars(Tbl_test_2,'Cn2');
Tbl_test_2 = removevars(Tbl_test_2,'mean_rain');
varNames_full_2 = varNames_full;
varNames_full_2(6)=[];
figure()
for i = 1:11

subplot(4,3,i)

hold on

toPlot=table2array(Tbl_test_2((Y_test_idx),i));
plot((toPlot),residVec,'.')
xlim([min(toPlot) max(toPlot)])
line(xlim(), [0,0], 'Linewidth', 0.5, 'Color', 'r');
xlabel((varNames_full_2{i}))
ylabel('Residual $Log_{10} C_{n}^{2}$','Interpreter','latex')

```

```
hold off  
end
```

Published with MATLAB® R2019b