AMERICAN UNIVERSITY OF BEIRUT

META-LEARNING TECHNIQUES FOR ASSESSING EEFLUX (METRIC) ET VERSUS REAL ET

by
SARA BASEM AWAD

A thesis
submitted in partial fulfillment of the requirements
for the degree of Master of Science
to the Department of Computer Science
of the Faculty of Arts and Sciences
at the American University of Beirut

Beirut, Lebanon
January 2021

AMERICAN UNIVERSITY OF BEIRUT

META-LEARNING TECHNIQUES FOR ASSESSING EEFLUX (METRIC) ET VERSUS REAL ET

by

SARA BASEM AWAD

Approved by:

Dr. Fatima Abu Salem, Associate Professor Department of Computer Science	Advisor
Dr. Hadi Jaafar, Associate Professor Department of Agriculture	Co-Advisor
Dr. Samer Kharroubi, Associate Professor Department of Nutrition and Food Science	Committee Member
Dr. Mohamed El Baker Nassar, Assistant Professor Department of Computer Science	Committee Member

Date of Thesis Defense: January 25, 2021

AMERICAN UNIVERSITY OF BEIRUT

THESIS, DISSERTATION, PROJECT RELEASE FORM

Student Name:	Awad	Sara	Basem
	Last	First	Middle

☑ Master's Thesis ○ Master's Project ○ Doctoral Dissertation

☐ I authorize the American University of Beirut to: (a) reproduce hard or electronic copies of my thesis, dissertation, or project; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

☑ I authorize the American University of Beirut, to: (a) reproduce hard or electronic copies of it; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes after: **One ___ year from the date of submission of my thesis, dissertation or project.**
Two ___ years from the date of submission of my thesis , dissertation or project.
Three ___ years from the date of submission of my thesis , dissertation or project.

February 4, 2021

Signature Date

Acknowledgements

I would like to express my deepest gratitude to my advisors Professor Fatima Abu Salem and Professor Hadi Jaafar for their guidance and support throughout my master's journey and to everyone who has supported me.

I am also grateful to the committee members who have been supportive throughout this process, in addition to their valuable discussions, ideas, and motivation.

It was a pleasure working closely with them where I have learned a lot by being introduced to the Agriculture Field and delving more into the Data Science and AI Fields which are a matter of my interest.

Finally, I would also like to thank my friends and family members who have shared this wonderful experience with me and for all the encouragement they have provided.

This accomplishment would not have been a success without you all.

Thank you.

An Abstract of the Thesis of

Sara Basem Awad for Master of Science
Major: Computer Science

Title: Meta-Learning Techniques for Assessing EEflux (METRIC) ET versus Real ET

Together with the increasing occurrence of drought events in many regions of the world, the constant need to increase agricultural production demands a more cautious regional water resources planning and assessment of irrigation needs and, thus, a more precise estimate of real evapotranspiration (ET). Several water management challenges have been addressed in recent years by models utilizing artificial intelligence. The main challenging aspects are represented by the choice of the best algorithm, availability of climatic data, and having adequate representative features. This study evaluated six machine learning models in two categories, i.e point-wise (Multi-Layer Perceptron (MLP), Ensemble of MLP, Meta-Learning), and probabilistic and uncertainty (Mixed Density Networks, MCDropout, Deep Ensemble) for accurately estimating daily ET with limited meteorological data in various climate regions (from dry continental to Mediterranean climates) and seasons from Ameriflux and Euroflux towers. Our datasets include a collection of publicly accessible remotely detected information traversing 26 sites from 2000 to 2018 such as Real ET values (the response variable) obtained from the Ameriflux and Euroflux towers, in addition to, climate and remotely-sensed data (LST, NDVI, and ALBEDO) obtained from EEflux.

In this thesis, we have incorporated utility-based learning and data oversampling (SMOGN) techniques targeting to enhance the recall of our models to capture extreme (relevant) values of ET. Furthermore, we have also experimented with different feature selection techniques and interpretability tools (SHAP and LIME) that show that air temperature, relative humidity, LST, and NDVI are the top contributing features. We have developed our study in a way to permit agricultural specialists and farmers to select between point-wise forecast, or probabilistic and uncertainty forecast. Our best performing point-wise model is a reptile meta learner that utilizes a multi-layer perceptron. Our meta-learner model achieved

an R2 of 0.79, RMSE of 0.90, and a Recall of 0.96 on the holdout subset of our entire dataset.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Background

Evapotranspiration (ET) is a metric that calculates the amount of water lost from the soil either by evaporation from the surface of the soil or through transpiration from the leaves of the plant. For water use and irrigation water management, accurate ET estimation is important to combat excessive water loss (Granata, 2019). To accurately estimate ET, researchers have developed several remote sensing techniques, but its estimation is considered to be a complicated process in which several meteorological variables are associated. This has inspired researchers to use machine learning to predict ET because of their ability to track complex relationships between dependent and independent variables. Traditionally, ET was measured using EEflux (Landsat-based evapotranspiration tool Earth Engine Evapotranspiration Flux (EEflux)), an automated calibration method initially based on METRIC, which uses the Landsat imagery archive stored on the Google Earth Engine (Foolad et al., 2018) to estimate EEFlux ET.

In this work, we explore the following research questions:

- Are the predictive models based on machine learning more accurate than the EEflux predictive models for ET?
- What machine learning systems can be used to outperform classic statistical approaches for predicting ET?
- Can we minimize the bias between the Real ET and EEflux (METRIC) ET?

To this end, we see our problem as a regression problem, capturing temporal and spatial variations in ET throughout the United States and Europe, and exploring several predictive frameworks that integrate machine learning models at their foundation, including vanilla pointwise prediction, Google Cloud's AutoML,

meta-learning for few-shot prediction, and probabilistic and uncertainty prediction. Our datasets consist of a series of publicly accessible remotely sensed data spanning 26 sites from 2000 to 2018, such as actual ET values (response variable) from the Ameriflux and Euroflux towers, as well as weather and remotely sensed data (LST, NDVI, and ALBEDO) from EEflux. Our data processing includes elements for unbalanced regression learning using utility-based regression techniques, and in our design, we strive to increase the recall of our models to capture extreme values of ET on excessively hot weather and that are most relevant to farmers. Our clustering of data across different climates and seasonality enables learning to be moved from higher air temperatures to lower air temperatures or from dry continental or Mediterranean climates (Csa, Dsa). Our Feature Selection Analysis helps us to further reduce the dimensionality of our problem in such a way as to require minimal feedback while retaining fair accuracy steps.

Our best performing model turned out to be the point-wise meta learner, using the Reptile algorithm based on a multi-layer perceptron (MLP – a class of feed-forward artificial neural network,). At the time of writing, this model achieved an R2 of 0.79, an RMSE of 0.90, and a Recall of 0.96 on the holdout subset of the entire dataset. This substantially improves on the results of a suite of statistical models on the same holdout dataset based on a simple linear regression model and mixed random effect models for raw and transformed data, a series of two-parts models (TPM) including TPM with normal regression, TPM with log-transformed data, and TPM with gamma regression and a log link. The latter suites achieved an R2 of 0.65 and an RMSE of 1.128 (credit to Dr. S. Kharroubi). Moreover, this same model enjoys a boost in performance across clusters upon the various climates, where R2 improves from 0.79 to 0.88, and RMSE improves from 0.90 to 0.50. This beats the baseline vanilla MLP by about 31.3% to 43.63% in R2 and by about 24.3% to 48.9% in RMSE, which confirms it is a highly skilled learner. This also beats Google Cloud Platform's autoML by about 8.2% to 8.6% in R2 and by about 10.8% to 48.9% in RMSE.

We have also trained our data using different probabilistic and uncertainty models. The best probabilistic model is Deep Ensemble. Deep Ensemble yielded an R2 score of 0.67, an RMSE of 1.15, and a Recall of 0.92 when trained on the entire dataset. Deep Ensemble witnessed a boost in performance across clusters upon the various climates, where R2 improves from to, and RMSE improves from to. Moreover, Deep Ensemble beats MC Dropout by about 0.58% to 1.36% in R2 and by about 0.68% to 0.94% in RMSE.

We conclude with a SHAP and LIME analysis for interpreting some of our best performing machine learning models, where it is shown that air temperature, relative humidity, LST, and NDVI are the top contributing features. SHAP also revealed across our models that high values of air temperature and NDVI drive

the models to predict high values of ET. On the other hand, SHAP revealed that low values of LST (land surface temperature), and relative humidity imply high values of predicted ET.

1.2 Overview

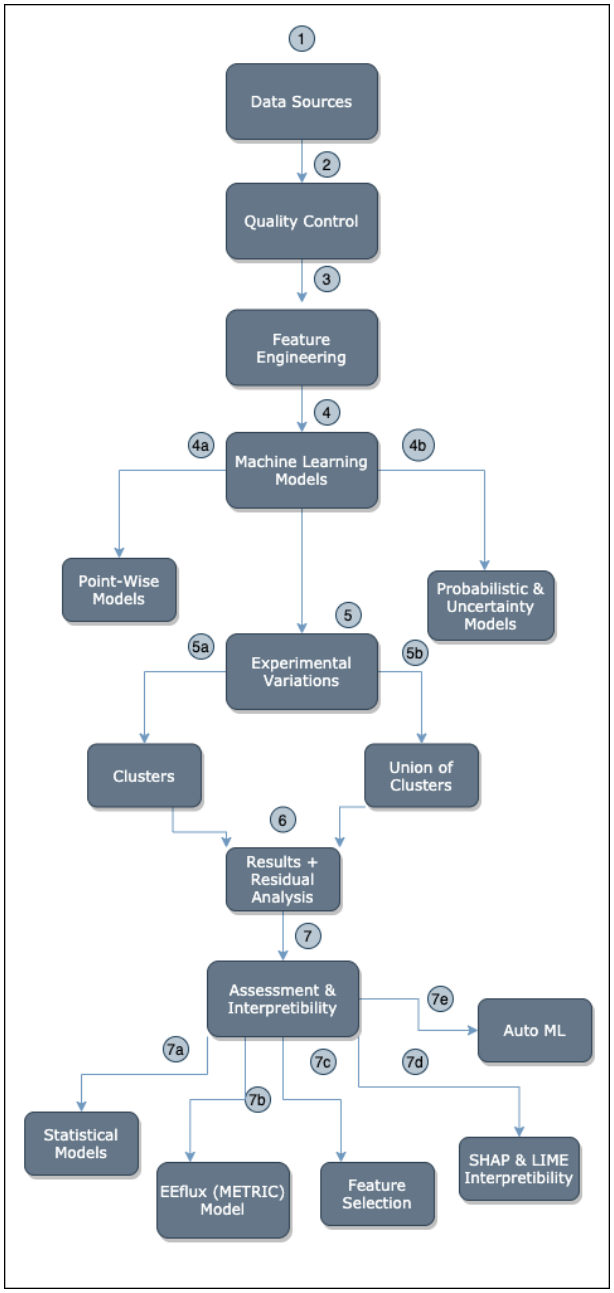


Figure 1.1: Report Flow

Chapter 2

Literature Review

In the work of (Baier & Robertson, 1965), daily Latent Evaporation is estimated from simple meteorological observations across six research areas in Canada. Meteorological observations were taken from May to October each year from 1953 to 1957. Simple and multiple linear correlation and regression were employed to study the impact of several meteorological and astronomical variables such as maximum temperature, temperature range, wind speed, vapor pressure deficit, solar energy, day length, total sky, solar energy on a horizontal surface, and duration of bright sunshine on the latent evaporation. The simple correlation shows that solar energy and sunshine were both more closely correlated with latent evaporation than temperature terms such as maximum and range. Furthermore, multiple regression analysis as well was employed to study the importance of the factors involved to develop equations for computing latent evapotranspiration. Results show that with only having minimum and maximum temperature and extraterrestrial radiation, the correlation coefficient was highly significant with R2=0.68. Adding one or more variables of solar energy, vapor pressure deficit, and wind speed results in the correlation coefficient ranging from R2=0.75 to R2=0.81. With all the six variables, results would improve to having R2=0.84.

On the other hand, the authors in (Granata, 2019) conducted a study to predict the actual ET at a Central Florida Site. The data recorded is between 28 September 2000 and 28 September 2004. Several machine learning algorithms (M5P Regression Tree, Bagging Random Forest, and Support Vector Regression) were employed on three different input combinations and different models. Model 1 consisted of sensible-heat flux, net solar radiation, moisture content of the soil, wind speed, mean relative humidity, and mean temperature. Model 2 consisted of net solar radiation, wind speed, mean relative humidity, and mean temperature. Model 3 consisted of net solar radiation, mean temperature, and mean relative humidity. A 10-fold cross-validation method was used where the data is being split randomly into 10 subsets with one set reserved as the validation data. The performance of the studied models was evaluated using these statistical indica-

tors: Nash-Sutcliffe model efficiency coefficient (NSE), Root mean squared error (RMSE), and Relative Absolute Error (RAE). This study showed that Model 1 yielded the best prediction for all the metrics scoring 0.987 for NSE, 0.14 mm/day for MAE, 0.179 for RMSE, and 15.4% for RAE for the best performing algorithm - Random forests. Model 2 and 3 showed quite analogous results but less satisfactory than Model 1. Future work would be to build a more powerful machine learning model for predicting the actual ET using the mean temperature, relative humidity, and net solar radiation with more complex models such as Artificial Neural Network (ANN) or Extreme Learning Machine (ELM).

Another work is done for estimating real ET is a study by (Huang et al., 2019). In this paper, the daily ET is estimated with limited meteorological data using a Categorical Boosting (CatBoost) algorithm and a gradient boosting decision tree. The result of the latter algorithm is compared to Support Vector Machine (SVM) and Random Forest (RF). The data used is a combination of meteorological data that includes both complete and incomplete combinations of solar radiation, relative humidity, maximum and minimum temperature, and wind speed from different weather stations during 2001–2015 in South China. The evaluation metrics used are RMSE, Mean Bias Error (MBE), Mean Absolute Percentage Deviation (MAPD), and R2 score. The study showed that when the complete combination of input exists, CatBoost reported the best accuracy, unlike for the other seven stations having incomplete input variables in which SVM reported the best accuracy scoring for RMSE (from 4.8% to 37.4%) and MAPE (-3.3% to 33.3%). Furthermore, the memory usage and computing time for processing data are much less for CatBoost which favors it to be a promising algorithm for ET0 estimation.

Nonetheless, the authors in (J. Fan et al., 2018) conducted a study on estimating the daily ET with limited meteorological data from 1961-2010 from eight representative weather stations in different climates in China. Four different input combinations were studied to evaluate their effectiveness on ET estimation. These input combinations consisted of daily minimum and maximum temperature, relative humidity, wind speed, global, and extraterrestrial solar radiation. A K-fold cross-validation method was used in which data was divided into five stages with four tree-based machine learning models including Random Forests (RF), M5 model tree (M5Tree), Gradient Boosting Decision Tree (GBDT), and eXtreme Gradient Boosting (XGBoost), Support Vector Machine (SVM), and Extreme Learning Machine (ELM). The performance of the studied models was evaluated using three statistical indicators: Coefficient of Determination (R2), Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). This study showed that SVM and ELM were the most performing and stable models scoring the least error when all the input variables were present. Furthermore, XGBoost and GBDT showed analogous results to those of Support vector machines (SVM)

and Extreme learning models (ELM) often exhibiting lower computational loss which favors them to be considered as an alternative technique for predicting ET. Results showed that in tropical and subtropical zones of China, global solar radiation was more important than the other variables. Further studies would include evaluating the given models in regions other than China and also rely on different time scales (hourly or monthly).

(Kaneko & Kennedy, 2019) introduced the concept of transfer learning from several regions when predicting real ET, in which they studied maize fields in different African countries by referring to publicly available remote sensing data. A long short-term memory (LSTM) based deep learning model was used to predict crop fields with a Gaussian Process Layer. The data was split into different splits: a random one and a chronological one. Using a random split, all models achieved high levels of accuracy, unlike a chronological one based on data quality. Moreover, their combined model, in which all the countries were trained collectively, showed that a collective deep learning model performs competitively with in-country models scoring 0.63 for R2. This aids the idea of learning from out-of-country features for countries with sparse data. Further studies could be using different algorithms such as a DNN model.

Since (Baier & Robertson, 1965) used no machine learning models, but rather the correlation is studied between several meteorological variables and latent evaporation, we were inspired to use Machine Learning to predict real ET. The work done by (Granata, 2019) has motivated us to experiment with several deep learning models along with probabilistic and uncertainty models i.e (Mixed Density Networks, Deep Ensemble. etc..) having different climatic conditions starting with air temperature, relative humidity, incoming shortwave radiation, etc... However, (Granata, 2019) only experimented with one site having a subtropical humid climate i.e. a warm and wet season from June to September and a mild dry season from October to May. We believe this restriction will not help in model generalization for unseen climate conditions, thus several sites and climatic regions should be considered. The studies by (Huang et al., 2019) and (J. Fan et al., 2018) have inspired us to predict real ET from data across several regions, and not be constricted to specific areas. We conducted a study on predicting daily ET during the years 2000-2019 for 26 stations collected from Ameriflux and Euroflux sites in different climate regions and vegetations. Our model is trained on different input features i.e wind speed, relative humidity, air temperature, land surface temperature, albedo, normalized difference vegetation index (NDVI), Site Id, Month, and Vegetation. We vary the aforementioned input features by including all stations, only Ameriflux stations, and only Euroflux stations and study the impact of each. We used a set of pointwise, probabilistic, and uncertainty models. The data is being split into training, validation, and testing data sets by 60%, 20%, and 20% respectively using a custom split by stations to ensure that

each station is included in all the data sets. The performance of our models was evaluated using several error metrics, correlation metrics, and utility-based metrics, some of which are as follows: Coefficient of Determination (R2), Root Mean Squared Error (RMSE), Mean Absolute Error: MAE, Recall, Precision, Accuracy, etc... Our work focuses on several regions as opposed to (W. X. L. X. Fan J. & Xiang, 2019) and we are using more complex models with less and different meteorological variables using filter-based approach feature selection methods. Our results reveal that the best machine learning model is when applying meta-learning using the Reptile algorithm showing (R2 of 0.79, RMSE of 0.9, and Recall of 0.96). We were also inspired by the work of (Kaneko & Kennedy, 2019) which led us to apply different clustering techniques. Our clustering of the data allows for transfer learning from sites with specific climates yielding better results than the union of clusters. As a result, we got high performing models trained on specific clusters better than when trained on the union of clusters, as opposed to low performing models trained on other clusters. Hence, these models trained on low performing clusters will learn from the ones excellently performing, and thus yield good results when the model is trained on a union of all clusters.

The report is divided into several chapters. **Chapter [3]** tackles the system and hardware setup being used, **Chapter [4]** tackles different data sources, **Chapter [5]** tackles different quality control measures, **Chapter [6]** and **Chapter [7]** tackles how data is being generated and transformed into a final coherent form. **Chapter [8]** tackles converting our regression problem into a classification-like problem using utility-based regression and up-sampling techniques. **Chapter [9]** lists several feature selection methods and scenarios applied to our data. **Chapter [10]** tackles different assessment metrics for evaluating our model's performance. **Chapter [11]** tackles applying better deep learning, generalization, and ensemble techniques. **Chapter [12]** states several experimental model setup. **Chapter [13]** tackles different point-wise models' history and implementation. **Chapter [14]** tackles the background of uncertainty quantification. **Chapter [15]** tackles different probabilistic and uncertainty models. **Chapter [16]** and **Chapter [17]** tackles different interpretability tools as being SHAP and LIME and the comparison between them.

Chapter 3

Systems and Hardware

3.1 Frameworks

A set of third-party libraries and frameworks are used in this project which are available on a central repository known as Python Package Index (PyPI). These frameworks are downloaded, installed by using a python tool, Preferred Installer Program (PIP), which is a command-line utility that aids in installing any PyPI package by issuing the following command: **pip install package-name**

Installing pip would vary depending on the operating system used:

- For mac users: sudo easy_install pip
- For Linux users: sudo apt-get install python3-pip

Some of the main libraries:

- Tensorflow 2: Offers easy model building, robust ML production, and strong experimentation tools
- Keras: Offers a set of off-the-shelf deep learning models
- Scikit-learn: Offers a set of off-the-shelf machine learning regression and classification models
- Pandas: Offers a set of tools for dataset handling and manipulation
- Pandas Profiling: Offers a set of tools for studying the data distribution and a handful set of visualizations
- Matplotlib: Offers a set of tools needed for plotting
- Rpy2: Offers easy connection between R and python

3.2 Installation

3.2.1 Google Cloud Service Setup

Install the Google Cloud SDK to be able to run core gcloud commands from the command-line. To do that, several steps should be done:

- Create a project on google cloud
- Python is required with versions 2.7.9 or higher. To check the Python version installed on your system

  `python -V` If nothing is displayed, python should be installed depending on the operating system being used from here
- Follow the steps in here to install gcloud and to initialize it

3.2.2 Correction Setup

- Download the fluxdataqaqc environment.yml file from here
- Create an environemnt in conda:

  ```
  conda env create -f environment.yml
  ```
- Activate the environment:

  ```
  conda activate fluxdataqaqc
  ```
- Install fluxdataqaqc in developer mode into your environment using

  ```
  pip package manager: pip install -e .
  ```
- To ensure that the library is working, run:

  ```
  from fluxdataqaqc import Data, QaQc, Plot
  ```

3.3 Configuration

3.3.1 Correction

A configuration file with a ".ini" extension should be created. This file should include METADATA & DATA sections. The data should either be in (.xlx and .xlsx) or comma-separated value (CSV) text file containing time series input data

having a column denoting the date and the time and a row indicating the variables names.
The structure of both sections includes a key mapped to a value (i.e key = value) which should contain the following information:

METADATA

It should include the following information:

- climate_file_path: The path of the xlsx or xsv data file
- station_elevation: The elevation of the station in meters
- station_longitude: The longitude of the station in decimal degrees
- station_latitude: The latitude of the station in decimal degrees
- site_id: The unique site identifier

DATA

It should include at least the following information:

- date_parser: The date format
- date: The name of the data column used
- R_n: The key mapping net radiation in the data file
- G: The key mapping ground flux in the data file
- LE: The key mapping latent flux in the data file
- H: The key mapping the sensible flux in the data file

3.4 Environment

The IDE being in use is the Anaconda Jupyter Notebook application which allows to create, display code chunks and mark up language sections, and being able to share generated reports with others. Code chunks are developed using Python which is included by default as part of the IDE.
We've further relied on several environments to run our scripts as such:

3.4.1 HPC Octopus Cluster

We used AUB's Octopus High-Performance Computing Clusters which offers strong computing power that is higher than a typical desktop and also offers GPU usages which we utilized when running TensorFlow models to shorten the model training time. Moreover, we also configure the RAM specifications, GPU model type, cores, etc.. and we simply submit our jobs for execution where it will be added to a queue of executable jobs. But there is only a certain number of available GPUs to run on which is a limitation to us in addition to the waiting queue process, which has motivated us to also rely on the Google Cloud platforms were by configuring our dedicated clusters.

3.4.2 Google Cloud Platform

Moreover, several Google Cloud Services were used to handle retrieving data, storing data, running and deploying machine learning models including the following:

- Compute Engine: Several virtual machines configured in different regions were set up to run an automated scraping script to retrieve data from EEflux Website
- Cloud Storage: Each virtual machine in the Compute Engine section is linked to a storage section i.e a bucket to store the satellite images being retrieved from EEflux and to store data sets from other websites
- DataProc Cluster: For each cluster, we set an initialization script to be able to use python 3 by default since the libraries we are using are compatible with this version. Furthermore, the jupyter notebook and Anaconda options are enabled to be able to use them to run our scripts using an interactive tool
- AI Platform: We are using this service to build machine learning models and compare our results to the ones obtained from Auto ML which runs internally a couple of models to find the optimal one for our problem having the option to predict an outcome using either an online or a batch prediction approach

Chapter 4

Study Area and Surveyed Sites

The data set was collected from Flux Towers (Ameriflux and Euroflux) across different climates and vegetations during the years 2000 till 2018 and their respective Landsat data is collected from the Earth Engine Evapotranspiration Flux website (EEflux). The data is collected from 26 sites from Ameriflux and Euroflux. We collected data from sites that represent different climatic zones and terrestrial vegetation types. Data from towers within the following climates were selected: Cfa (Humid Subtropical), Dsa (Dry Continental), Csa (Mediterranean), Csb (Mediterranean), and Cwa (Humid Subtropical). Site vegetation types include Grasslands (GRA), Crop Lands (CRO), Closed Shrublands, Evergreen Broadleaf Forests (EBF), and Evergreen Needle Leaf Forests (ENF).

4.1 Ameriflux

Ameriflux, one of the DOE Office of Biological and Environmental Research's (BER), was launched in 1996. It contains information for more than 50 sites. This website tracks several sites' information such as its variables, years it is collected, geographic location, etc... Data is available on a half-hourly, hourly, daily, weekly or monthly basis. (*Search AmeriFlux Sites*, 2019)

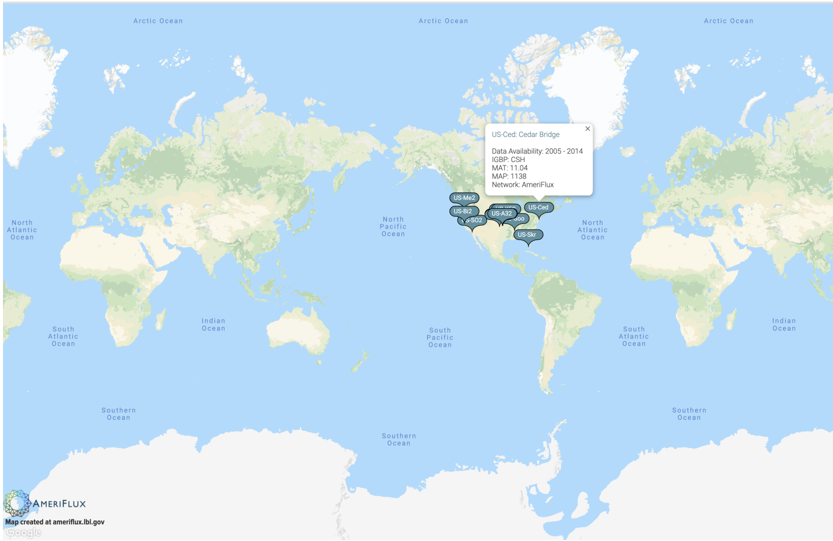


Figure 4.1: Sample Sites from Ameriflux

4.1.1 Methodology

The following procedure is performed in order to retrieve each site's information:

- First, a list of all existing sites is obtained with different climatic weathers, vegetation, and the years the data is collected for each and their geographic location **Table** [**4.1**]
- Then, a list of flux and auxiliary variable for each site is retrieved
- Then, each site's information along with its variables is exported into a CSV file tracking each variable on a half-hourly basis

To retrieve the elevation in meters per site the following is done:

- Go to `https://www.freemaptools.com/elevation-finder.htm`
- Select the latitude and longitude option from the selector
- Provide the site's latitude and longitude and press enter which will result in displaying the elevation in the map in meters
- Go to `https://www.maps.ie/coordinates.html` and provide the site's latitude and longitude to validate the elevation being retrieved above

4.1.2 Sites Details

Several sites with different climatic weathers and vegetation are studied:
Climates:

Table 4.1: Ameriflux Sites

Site Id	Latitude	Longitude	Years	Climate	Vegetation
US-AR1	36.4267	-99.42	2009 - 2019	Dsa	GRA
US-AR2	36.6358	-99.5975	2009 - 2012	Dsa	GRA
US-A32	36.819268	-97.819772	2015 - 2019	Cfa	GRA
US-A74	36.808464	-97.548854	2016 - 2019	Cfa	GRA
US-Bi2	38.109	-121.535	2017 - 2019	Csa	CRO
US-Ced	39.8379	-74.3791	2005 - 2019	Cfa	CSH
US-Kon	39.0824	-96.5603	2006 - 2019	Cfa	GRA
US-Pon	36.76667	-97.13333	1997 - 2001	Cfa	CRO
US-Shd	36.93333	-96.68333	1997 - 2001	Cfa	GRA
US-Skr	25.362933	-81.077582	2004 - 2019	Cwa	EBF
US-Snd	38.0373	-121.7537	2007 - 2015	Csa	GRA
US-SO2	33.3738	-116.6228	1997 - 2019	Csa	CSH
US-SP2	29.7648	-82.24482	1998 - 2008	Cfa	ENF
US-Twt	38.1087204	-121.6531	2009 - 2019	Csa	CRO
US-Tw2	38.1047	-121.6433	2012 - 2013	Csa	CRO
US-Var	38.4133	-120.9507	2000 - 2009	Csa	GRA
US-Wlr	37.5208	-96.855	2001 - 2004	Cfa	GRA

- Cfa (Humid Subtropical): Mild with no dry season and hot summer
- Dsa (Dry Continental): Hot summer
- Csa (Mediterranean): Mild with dry and hot summer
- Csb (Mediterranean): Mild with dry and warm summer
- Cwa (Humid Subtropical): Dry winter and hot summer

Vegetation/IGBP:

- Grasslands (GRA): Lands with herbaceous types of cover. It has permanent combination of water and vegetation having the vegetation present in salt, fresh water, or brackish
- Crop Lands (CRO): Lands that are covered with temporary crops followed by a harvest then a bare soil period
- Closed Shrublands (CSH): Lands having woody vegetation with shrub canopy cover and less than 2 meters tall
- Evergreen Broadleaf Forests (EBF): Lands that are dominated with woody vegetation having a percent cover $> 60\%$ and height exceeding 2 meters

- Evergreen Needle Leaf Forests (ENF): Lands that are dominated with woody vegetation having a percent cover > 60% and height exceeding 2 meters

For more information about each IGBP check here

4.2 Euroflux

European Fluxes Database Cluster: A database that hosts several flux measurements such as meteorological variables, ancillary data, and meta-information from different sites in and outside Europe. The data is represented in 3 different levels. Each level has a different format of representing the data. In each "-9999" indicates a missing value

4.2.1 Methodology

The following procedure is performed in order to retrieve and group the data:

- Request data from here for sites having IGBP code as CRO to retrieve cropland sites
- Run a script that first extracts all the zip files containing the meteorological variables
- Group each sites information for all the years in one sheet on a half-hourly format

4.2.2 Sites Details

Several sites with different climatic weathers and vegetation are studied:

Table 4.2: Euroflux Sites

Site Id	Description	Latitude	Longitude	Years	Vegetation
BE-Lon	Lonzee	50.5516198	4.7462339	2004 - 2018	CRO
CH-Oe2	Oensingen crop	47.286417	7.73375	2004 - 2015	CRO
DE-Geb	Gebesee	51.09973	10.91463	2001 - 2019	CRO
DE-Kli	Klingenberg	50.89306	13.52238	2004 - 2018	CRO
DE-RuS	Selhausen Juelich	50.86590702	6.447144704	2011 - 2018	CRO
DE-Seh	Selhausen	50.8706233	6.44965306	2007 - 2010	CRO
DK-Fou	Foulum	56.484199	9.5872201	2004 - 2005	CRO
DK-Ris	Risbyholm	55.53027778	12.09722222	2004 - 2005	CRO
ES-ES2	El Saler-Sueca (Valencia)	39.27555556	-0.315277778	2004 - 2010	CRO
FI-Jok	Jokioinen	60.8986	23.51345	2000 - 2003	CRO
FR-Avi	Avignon	43.91608333	4.878055556	2004 - 2006	CRO
FR-Gri	Grignon	48.84422	1.95191	2004 - 2019	CRO
IE-Ca1	Carlow crop	52.85879167	-6.918136111	2004 - 2008	CRO
IT-BCi	Borgo Cioffi	40.52375	14.95744444	2004 - 2015	CRO
IT-CA2	Castel d'Asso2	42.37721944	12.02603889	2011 - 2014	CRO
IT-Cas	Castellaro	45.07004722	8.717522222	2006 - 2010	CRO
IT-Ro3	Roccarespampani3	42.37539	11.91542	2007 - 2013	CRO
IT-Ro4	Roccarespampani4	42.37333	11.91922	2007 - 2013	CRO
NL-Lan	Langerak	51.95360184	4.902900219	2005 - 2006	CRO
NL-Lut	Lutjewad	53.39892222	6.356027778	2006 - 2007	CRO
NL-Mol	Molenweg	51.65	4.63908333	2005 - 2006	CRO
UK-ESa	East Saltoun	55.9069444	-2.85861111	2003 - 2006	CRO
UK-Her	Hertfordshire	51.78379822	-0.47608	2006	CRO

4.3 EEflux

EEflux, Earth Engine Evapotranspiration Flux, is based on the METRIC model (ref), operating on Google Earth's Engine System. EEflux generates on-demand ET estimations for Landsats (5,7 or 8) scenes. This website can process Landsat scenes from 1984 till the present date in almost every land area on the Globe at 30 m resolution. It has an interactive temporal and spatial search GUI engine in which a geographic location and date interval is specified to download a geo-tiff satellite image. Only one satellite image can be viewed or downloaded at a time, this has urged us to build an automated scraper to download multiple satellite images for a specified site, a date interval, and a metric without providing any

manual intervention ("EEFlux: A Landsat-based Evapotranspiration mapping tool on the Google Earth Engine", 2015).

4.3.1 Methodology

We have built an automated scraper in python that was fed a set of attributes(latitude, longitude, date interval) to retrieve a certain metric (EEflux ET, EEflux ETr, EEflux ETo, or EEflux ETrf). We have run the scraper on different parallel clusters on the google cloud compute engine to optimize the performance of the scraping process and thus faster results. We thus further obtained a model output for each satellite overpass and then extracted the mean values for nine pixels including the tower site (i.e. a 45m buffer around the tower) as well as the % cloud cover (to discard cloudy pixels). We have retrieved actual ET, Reference ET (Alfafa and grass), and Evaporative fraction, Albedo, NDVI, and Landsurface temperature data for the 26 sites obtained above across the years 2000-2018.
The following is performed to retrieve satellite images for the mentioned sites:

1. Build an automated scraper in python that was fed a set of attributes to retrieve data accordingly. Attributes provided were:

    - Latitude: The latitude of the site
    - Longitude: The longitude of the site
    - Date Info: A range of start and an end date filter with the format "YYYY-MM-dd"
    - Type_id:
        - 1 is used for the "eta" parameter which will get the records for "ACTUAL ET"
        - 2 is used for "etr" parameter which will get the records for "ALFAFA REFERENCE ET (ETr)"
        - 3 is used for "eto" parameter which will get the records for "GRASS REFERENCE ET (ETo)"
        - 4 is used for "etrf" parameter which will get the records for "FRACTION REFERENCE ET (ETrF)"
    - Metric/Type_Name: A metric or a data layer to download a satellite image having the following options: "ACTUAL ET", "ALFAFA REFERENCE ET (ETr)", "GRASS REFERENCE ET (ETo)", & "FRACTION REFERENCE ET (ETrF)"

2. After having to select the set of attributes, a list of zip files is downloaded. Each zip file is named according to the satellite image followed by the metric and the date on which it was captured on

3. All files are unzipped to extract the "geo-tiff" files representing the actual image which has a % cloud cover selection

4. The metric value (for instance Actual ET) is then extracted from each image and then the mean of that metric is calculated based on the 8-neighbor pixels of the current site which will be useful in case the metric value was not available for a certain date.

5. Each metric value will then be exported into a CSV file with its respective date and % cloud cover and its mean value

The above methodology is also explained in details in **Algorithm [1]**

Algorithm 1 EEflux Scraping

$BUCKET\_NAME$ = "bucketname"
df ← Read("gs:// + BUCKET_NAME + /Data sets")
site_id ="US-Kon"
type_id = 1
site_df = *df*['Site Id" == site_id]
site_df["*date_info*"] = *site_df*["start_date"] + "to" + *site_df*['end_date']
response ← Call post api with *site_df*["*date_info*"], *site_df*["*lat*"] and *site_df*["*long*"]
data = to_dict(*response*)
response_df ← Create(*data*['Date'], *data*['Tier'], *data*['id'], *data*['Cloud'])
type_name ← get(*type_id*)
for row *in response_df* **do**
    raster_url ← Call post api with *row*['*id*'] and *type_name*
    $BUCKET\_NAME$ ← Upload(*raster_url*)
    raster ← Unzip file from *raster_url*
    metric_value ← Extract *metric_value* from *raster*
    site_df["type_name'] = *metric_value*
    site_df["type_name_mean'] = mean(*metric_value*)
    Del *raster*
end for
Export *site_df* to $BUCKET\_NAME$

4.4 Reference Evapotranspiration

The Reference Evapotranspiration is an estimation of evapotranspiration from the surface reference and is also known as reference crop transpiration. There are two types of surfaces: the standardized reference evapotranspiration for the short crop with a height of around 0.12 m which is denoted by ET_o and the standardized reference evapotranspiration for the tall crop with a height of around 0.50

m which is denoted by ET_r. ET_o is similar to the clipped, cool-season grass, and ET_r is similar to the full-cover alfalfa (R. G. Allen et al., 2005). The reference evapotranspiration was introduced to study the behavior of the atmosphere regardless of the crop type, and development. Furthermore, ET_o that are measured at different seasons or locations are considered comparable as they both denote to ET from the same reference surface being affected by the climate parameters, therefore ET_o can be computed from the weather data and is often a climatic parameter.

It can be estimated either from the meteorological data or from the pan evaporation. FAO Penman-Monteith is considered the recommended method for calculating ET_o which uses meteorological data (i.e radiation, air humidity, air temperature, and wind speed data) even when the climatic data are missing (*Chapter 4 - Determination of ETo*, 2019a). On the other hand, the pan evaporation method is also used to estimate ET_o using coefficients to relate the pan evaporation to ET_o.
In our case, the refET library will be used to compute the daily reference ET following the ASCE Standardized Reference Evapotranspiration Equations for both ET_r and ET_o (R. G. Allen, 2005).

4.4.1 Methodology

We will compute ET_r and ET_o for all the sites on a daily basis as being stated by the below algorithm

Algorithm 2 Reference Evapotranspiration Calculation

1: $ea = \frac{RH}{100 \times es}$
2: tmean $\leftarrow$ avg(TA)
3: $es = 0.6108 \times exp(\frac{17.27 \times tmean}{tmean + 237.3)})$
4: $rs =$ SW_IN
5: **if** WS is NaN **then**
6: $uz = 2$
7: **else**
8: $uz =$ WS
9: **end if**
10: **if** ET_r **then**
11: $etr \leftarrow$ refET.Daily(min_TA, max_TA, ea, rs, uz, zw, $elevation$, $latitude$, doy, method='asce').etr()
12: $ETrF\_bowen = LE\_bowen\_corr(mm)$ / ET_r
13: **else if** ET_o **then**
14: $eto \leftarrow$ refET.Daily(min_TA, max_TA, ea, rs, uz, zw, $elevation$, $latitude$, doy, method='asce').eto()
15: $ETrF\_bowen = LE\_bowen\_corr(mm)$ / ET_o
16: **end if**

Detailed information about this algorithm is described in Ref ET

After having computed ET_r and ET_o which is what we're studying, we will be retrieving their equivalent ET_r and ET_o from EEflux or Climate Engine using the automated engine scraper in order to compare each.

4.5 Climate Engine

Climate Engine is a web application powered by Google Earth Engine which is used to analyze and visualize gridded weather data and satellite observations for decision support related to agriculture, ecology, etc... It supports on-demand cloud computing of remote sensing data and visualization of climate. We will use this tool to extract several values in addition to what we're retrieving from EEflux for Ameriflux and Euroflux sites.

4.5.1 Methodology

To retrieve the ET_o and ET_r values for Ameriflux sites, the following is done:

- Set the Type field to Climate/Hydrology
- Set the Data field to METADATA/gridMet
- Set the variable to one of the following:
  - ASCE Grass Reference Evapotranspiration (ET_o)
  - ASCE Alfalfa Reference Evapotranspiration (ET_r)
- Set the time period to a custom date range option to vary the start and end dates depending on the available data per site
- Set the region to the point option to be able to add the latitude and longitude per site
- Click the get map layer to process the request
- After the data is processed and retrieved, it is downloaded in a CSV file format

Since ET_r is not available for Euroflux sites, we will be using the potential evaporation instead by doing the following:

- Set the Type field to Climate/Hydrology
- Set the Data field to CFS Reanalysis
- Set the variable to one of the following:
  - Potential Evaporation
- Set the time period to a custom date range option to vary the start and end dates depending on the available data per site
- Set the region to the point option to be able to add the latitude and longitude per site
- Click the get map layer to process the request
- After the data is processed and retrieved, it is download in a CSV file

The web application is available in here

Chapter 5

Quality Control

5.1 Data Cleaning

After data is collected from different sources, it further needs to be cleaned and corrected.

5.1.1 Ameriflux and Euroflux

- Convert the occurrences of "-9999" with NaN which better indicates a missing value
- Remove missing LE & its variants
- If more than one variant of a certain column exist, take only the first variant
- Remove data with malformed dates i.e having days or months that do not exist
- There were negative LE values as well as very high LE values. We thus limited the ranges of our LE variable to values between 1 and 15 mm

5.1.2 EEflux

- Impute the missing records for $ModeledET$ to be the Mean Modeled ET
- Impute the missing records for ET_r to be the Mean Modeled Reference ET_r
- Impute the missing records for ET_o to be the Mean Modeled Reference ET_o
- Impute the missing records for ET_rF to be the Mean Modeled Reference ET_rF

- We substituted values of $ModeledET$ (EEflux ET) that are less than 0.3 to be 0.3 and values greater than 15 to be 15. The reason for choosing these values to substitute with is that we have performed a set of experiments with several cutoffs varying between 0, 0.1,0.2,0.3,0.4,0.5, and 1. Our purpose for that since we do not want to tamper with a big percentage of the data. Thus, we checked the percentage of data with values less than the latter lower bounds. The lowest percentage was observed with the lower bound 0.3 and below. Thus, we performed all our residual analysis experiments with EEflux ET having the lower bound ranges 0.1,0.2,0.3. The best results were for 0.3, hence we decided on it.

- We will correct the rows that are redundant i.e (having the same values) in their input features but only differ in the cloud cover percentage column. We will only correct one row and we will further discard the others and consider them as erroneous. To decide on which row to correct and how to correct it, we will have to rely on real ET and EEflux ET as such:
  We compute the difference in EEflux ET values between the redundant rows.

  - If the difference was less than 0.3, we will further compute the average for EEflux ET between these rows and discard all the others.
  - If the difference was higher than the predefined threshold, then we will compute the difference between EEflux ET and real ET for each redundant row. We will keep the row that shows the least discrepancy (lowest value) and we will discard the others.

5.2 Data Correction

It is often vital to ensure that the hydrometeorological measurements & turbulent fluxes are accurate and to quantify any systematic errors or uncertainties which play an important role in irrigation and energy management. Eddy covariance (EC), which is used to determine the exchange of water vapor between an ecosystem and the atmosphere, leads to an underestimation of sensible heat (H) and latent heat (LE) fluxes by causing non-closure of the surface energy balance models to estimate them i.e the turbulent heat fluxes are smaller than the available energy which is the sum of the ground heat flux and net radiation at the surface (5.1). Several methods have been proposed to correct these fluxes to better close the energy balance model. We have computed the corrections using two approaches: one is applying the Bowens Ratio by (Mauder, 2013). and the other using a python package library flux-data-qaqc which performs two corrections: the Energy Balance Ratio method, & a variant of the Bowens Ratio method.

$$R_n - G = H + LE \tag{5.1}$$

5.2.1 Bowen's Ratio

The method proposed by (Mauder, 2013) relies on preserving the Bowen's ratio (H/LE) on a half-hourly basis before and after the correction is applied. This method doesn't cause the individual flux points (i.e half-hourly) to be perfectly close but does for the daily time scale.

$$C = \frac{\sum_{i=1}^{k} H_i + LE_i}{\sum_{i=1}^{k} R_{n,i} + G_i} \tag{5.2}$$

$$H_c = \frac{H_m}{C} \tag{5.3}$$

$$LE_c = \frac{LE_m}{C} \tag{5.4}$$

Methodology

1. Remove all records having $R_n < 20$ which are close to sunset
2. Calculate the correction factor for each day depending on the valid k (half-hourly) records (eq:5.2)
3. Compute the corrected H (eq:5.3)
4. Compute the corrected LE (eq:5.4)
5. Compute the residual energy balance after the correction to quantify the difference after correction is applied (eq:5.1)

5.2.2 Bowen's Ratio - FluxQAQC

The Bowen Ratio energy balance closure method is performed on a daily basis by preserving the Bowens ratio (H/LE). This correction closes the energy balance exactly, when the balance is evaluated on a daily time scale (i.e, daily sums), but not when evaluated on flux data points (e.g., 30-minutes). This is implemented by authors in flux-data-qaqc.

Methodology

1. Compute the Bowens Ratio β

$$\beta = \frac{H}{LE} \tag{5.5}$$

2. Compute the corrected LE

$$LE_{corr} = \frac{(Rn - G)}{(1 + \beta)} \tag{5.6}$$

3. Compute the corrected H

$$H_{corr} = LE_{corr} \times \beta \tag{5.7}$$

The variables names which are computed using the above approach are as follows:

- br: Bowen Ratio
- ebr: The energy balance ratio
- ebr_corr: The corrected energy balance ratio
- LE_corr: The corrected Latent energy flux
- H_corr: The corrected sensible heat flux
- ET: The evaportanspiration
- ET_corr: The corrected evapotranspiration
- $energy$: The energy that is computed from R_n - G
- $flux$: The flux which is computed from LE + H
- $flux\_cor$: LE_corr + H_corr

5.2.3 Energy Balanced Ratio (EBR)

The energy balance method constitutes removing extreme values of the daily time-series data and gap-filling it. Then the inverse of this data is used as a series of the correction factor for the initial latent energy (LE) and sensible heat (H) time series flux data. (Mauder, 2013)

1. Filter out poor quality data if quality control flags are provided with the data set
   E.g.: $LE\_F\_MDS\_QC$ is a QC field for a gap filled LE

2. Compute the energy balance ratio daily from the raw data

$$EBR = \frac{H + LE}{Rn–G} \tag{5.8}$$

3. Filter out EBR values that are outside 1.5 times the interquartile range

4. For each day, a sliding window of +/- 7 days (15 days) is used in order to select up to 15 values

5. For each day, compute the percentile (defaults to 50) of the 15 EBR values, then check if the inverse of the EBR is > |2| or if it's inverse multiplied by LE would result in a flux greater than 850 or less than -100 w/m^2, if that happens then the gap filling procedure is left for later

6. If less than +/- 5 days exists in the 15 sliding window day, then compute the mean EBR for all the days in a +/- 5 day (11 day) sliding window and then apply the criteria for EBR mentioned in the previous step

7. If no ERB data is available in the +/- 5 sliding window to average, then fill the remaining gaps of EBR with the mean from a +/- 5 sliding window over the day of the year mean for all the years on record i.e 5 day climatology. Then compute the 5 day climatology from the filtered EBR and then apply the criteria for EBR mentioned in the previous step

8.
   - Use the filtered EBR time series data to correct LE and H:

$$EBC_{CF} = \frac{1}{EBR} \tag{5.9}$$

$$LE_{corr} = LE \times EBC_{CF} \tag{5.10}$$

$$H_{corr} = H \times EBC_{CF} \tag{5.11}$$

   - Then use the corrected turbulent fluxes to calculate the corrected EBR:

$$EBR_{corr} = \frac{H_{corr} + LE_{corr}}{Rn - G} \tag{5.12}$$

   - Compute ET from LE using the average air temperature to adjust the latent heat of vaporization

$$ET = 86400_{sec \cdot day^{-1}} \times \frac{LE}{2501000_{MJ \cdot kg^{-1}} - (2361 \cdot T_C)} \tag{5.13}$$

     having evapotransipiration (ET) in $mm \cdot day^{-1}$
     LE is latent energy flux in $w \cdot m^{-2}$
     T is air temperature in degrees Celsius

9. Optional Step: Apply gap filling procedure to ET using gridMET reference ET by downloading ET_r, the overlapping gridMET cell (The site must be in CONUS) and then computing reference ET fraction ET_{rF} as follows:

   - Remove outliers i.e values outside 1.5 times the interquartile range
   - Data is then smoothed with a 7 day moving average (i.e a minimum of 2 days must exist in the window)
   - Linear interpolation is applied to fill the remaining gaps

The variable names which are computed using the above approach are as follows:

- LE_corr: The corrected latent energy flux
- H_corr: The corrected sensible heat flux

- *ebr*: The energy balance ratio
- *ebr_corr*: The corrected energy balance ratio
- *ET*: The evaportanspiration
- *ET_corr*: The corrected evapotranspiration
- *ebr_corr*: The inverse of energy balance closure correction factor
- *ebr_5day_clim*: A 5 day climatology of the filtered Energy Balance Ratio
- $gridMET\_ET_r$: Is the gridMET alfalfa reference *ET* (nearest cell)
- ET_r: The reference evaportanspiration
- ET_{rF}: The fraction reference evaportanspiration for *ET_corr* which is computed by applying: *ET_corr* / $gridMET\_ETr$

Chapter 6

Data Generation

After cleaning and analyzing the Ameriflux & Euroflux data sets, we will need to correct each data set and then generate the new ones which will then be joined with their respective EEflux data. To do so, we are using two approaches to correct our data, the manual or the library correction generation method.

This section focuses on re-sampling the data from hourly to daily after having done some cleaning and analysis of our data sets. Newly generated data will be produced as a result of correcting the data, reference evapotranspiration generation, and joining the Tower data with its respective EEflux data. We are using two approaches to correct our data, the manual and the library correction methods. Each method is described in details in **Chapter [5]**

6.1 Manual Data Generation

This section focuses on performing all the necessary steps to produce a daily corrected data set and another that is joint with EEflux using the manual bowen correction. Steps involved are:

1. Global Variable Initialization
2. Data Cleaning
3. Data Processing
4. Data Aggregation & Correction
5. Reference Evapotranspiration Calculation
6. EEflux Joining with Tower Data

6.1.1 Global Variables Initialization

This section focuses on setting and initializing the global variables which are used in the below procedure

- Set BASE_PATH: "Users/saraawad/Desktop/Datasets/Google/"
- Set INPUT_PATH: "/Users/saraawad/Hourly Cleaned/Ameriflux/"
- Set FLUX_PATH: "/Users/saraawad/Desktop/flux-data-qaqc/sites/data"
- Set FLUX_PROCESSED_HOURLY_PATH: BASE_PATH + "Common/Manual/Hourly/"
- Set FLUX_DAILY_PATH: BASE_PATH + "Common/Manual/Daily/"
- Set FLUX_PROCESSED_DAILY_PATH: BASE_PATH +"Ref/"
- Set AMERIFLUX_TA: BASE_PATH + "Ameriflux_TA.csv"
- Set EUROFLUX_TA: BASE_PATH + "Euroflux_TA.csv"
- Set EEFLUX_ET_PATH: BASE_PATH + "EEflux/EEflux_sites.csv"
- Set EUROFLUX_ETo_PATH: BASE_PATH + "Euroflux_ETo_grass.csv"
- Set EUROFLUX_ETr_PATH: BASE_PATH + "Euroflux_Potential_ET.csv"
- Set AMERIFLUX_ETo: BASE_PATH + "Ameriflux_ETo_grass.csv"
- Set AMERIFLUX_ETr: BASE_PATH + "Ameriflux_ETr_alfafa.csv"
- Set EEFLUX_ALBEDO_NDVI: BASE_PATH + "EEflux/EEflux_Albedo_NDVI.csv"
- Set FLUX_JOINT_PATH: BASE_PATH + "Common/Manual/Joint"

6.1.2 Data Cleaning

This section focuses on cleaning the data set and removing NaN values

1. Load each Ameriflux hourly data from the **INPUT_PATH**
2. Process each data set by doing the following:
   (a) Drop invalid columns i.e ones having the suffix "_SSITC_TEST" or having the prefix "WS_MAX"
   (b) Drop rows having NaN values
   (c) Drop all the input column variants ("NETRAD", "H", "G", "RH", "WS", "TA") and use only the first variant
   (d) Export the processed data sets for each site to **FLUX_PATH**
   (e) Add Euroflux hourly data sets to path **FLUX_PATH**

6.1.3 Data Processing

This section focuses on preparing the data set for the correction by computing columns on the hourly data set and using them when re-sampling the data set to daily

1. Loop over all the unique sites from sheet "filtered_sites_encoded_all.xlsx"
2. Perform a binary encoding to the Site Id, Vegetation, Climate, and Month columns as mentioned in **Chapter [5]**
3. Add columns from the filtered sheet to the current data set
4. Generate date components fields i.e Year, Month, Day columns
5. Check if all the Bowens elements exist i.e ("H", "LE", "NETRAD", "G"), if so start computing the Bowens correction otherwise exit and do nothing
6. Add a new column "LATENT_SENSIBLE" by summing up "H" + "LE" when "NETRAD" $>$ 20 otherwise set to 0
7. Add a new column "NETRAD_SOIL" by summing up "NETRAD" + "G" when "NETRAD" $>$ 20 otherwise set to 0
8. Export each processed data set for each site to path **FLUX_PROCESSED_HOURLY_PATH** with the site id suffixed with"_Hourly.csv" for instance: US-Var_Hourly.csv
9. Concatenate all data sets into one data set
10. Export the concatenated data set to path **FLUX_PROCESSED_HOURLY_PATH** with the name "all_hourly.csv"

Algorithm 3 Manual Bowen Cleaning and Processing

Data:

Load hourly data from **INPUT_PATH** into *df_hourly*
Load sites data from sheet **filtered_sites_encoded_all** into *sites_df*

Del columns with suffix "_SSITC_TEST" & "WS_MAX"
Del NaN rows
Del columns with suffix "_1", "_2", "_3"
Export *df_hourly_cl* into **FLUX_PATH**
for *site in sites_df* **do**
  Binary encode "Site_Id", "Vegetation", "Climate", & "Month"
  Compute "Year", "Month", & "Day" from "Date" column
  Set $BOWENS\_ELEMENTS$ = ["H", "LE", "NETRAD", "G"]
  if $BOWENS\_ELEMENTS$ exists **then**
    if "NETRAD" > 20 **then**
      $LATENT\_SENSIBLE \leftarrow$ sum("H", "LE")
      $NETRAD\_SOIL \leftarrow$ sum("NETRAD", "G")
    else
      $LATENT\_SENSIBLE = 0$
      $NETRAD\_SOIL = 0$
    end if
  else
    continue
  end if
  Export *df_hourly_cl* into **FLUX_PROCESSED_HOURLY_PATH**
  Add site to sites
end for
Export sites to **FLUX_PROCESSED_HOURLY_PATH**

6.1.4 Data Aggregation and Correction

This section focuses on re-sampling the data from an hourly to daily basis by aggregating the correction elements columns and other main columns

1. Load the data sets from path: **FLUX_PROCESSED_HOURLY_PATH**

2. Re-sample the hourly data to daily by aggregating all the main variables and summing up"LATENT_SENSIBLE" and "NETRAD_SOIL" and grouping by the "Site Id" and "Date" columns

3. Generate date components fields i.e Year, Month, Day columns

4. Add a new column "C_BOWENS" by dividing "LATENT_SENSIBLE" with "NETRAD_SOIL"

5. Add a new column “LE_bowen_corr” which is set to “LE” divided by “C_BOWENS” when “C_BOWENS” > 0 otherwise set to NaN

6. Add a new column “H_bowen_corr” which is set to “H” divided by “C_BOWENS” when “C_BOWENS” > 0 otherwise set to NaN

7. Drop a row having one of “LE_bowen_corr” and “H_bowen_corr” as NaNs

8. Drop“LATENT_SENSIBLE” and “NETRAD_SOIL” columns

9. Generate lags for the columns: “H”, “H_bowen_corr”, “G”, “NETRAD”, “WS”, “RH”, “TA” as mentioned in **Algorithm** [**8**]

10. Calibrate LE and H by adding a conversion rate to each of “LE_bowen_corr” and“H_bowen_corr” i.e dividing each by 28.94. The resulting columns will have a suffix (mm)

11. Order columns following the below criteria

    (a) Columns coming from the filtered sheet i.e Date, Site Id, Year, Month, Day, Climate, Vegetation, Latitude, Longitude, Elevation(m)

    (b) Weather columns i.e WS, RH, TA, NETRAD, SW_IN followed by their lags i.e WS-1, WS-2,... WS-5, RH-1, RH-2,... RH-5, etc...

    (c) Energy balanced columns i.e G, H, H_bowen_corr, H_bowen_corr(mm), LE, LE_bowen_corr, LE_bowen_corr(mm)

    (d) Reference ET columns i.e ETo, ETr, ETof_bowen, ETrf_bowen

    (e) EEflux columns if exists i.e EEflux Albedo, EEflux NDVI, EEflux LST, EEflux ET, Cloud Cover, EEflux ETo, EEflux ETr

12. Export each daily data set for each site to path **FLUX_DAILY_PATH** with the site id suffixed with“_Daily.csv” for instance: US-Var_Daily.csv

13. Concatenate all data sets into one data set

14. Export the concatenated data set to path **FLUX_DAILY_PATH** with the name “all_daily.csv”

Algorithm 4 Manual Bowen Aggregation and Correction

Data:

Load data from sheet **FLUX_PROCESSED_HOURLY_PATH** into *df_hourly_pr*

for *df_site in df_hourly_pr* **do**
  Group each *df_site* by *SiteId* & *Date*
  df_site is re-sampled to *df_site_daily*
  $LATENT\_SENSIBLE \leftarrow \text{sum}(LATENT\_SENSIBLE)$
  $NETRAD\_SOIL \leftarrow \text{sum}(NETRAD\_SOIL)$
  $C\_BOWENS = LATENT\_SENSIBLE \ / \ NETRAD\_SOIL$
  if "C_BOWENS" > 0 **then**
    $LE\_bowen\_corr = LE \ / \ C$
    $H\_bowen\_corr = H \ / \ C$
  else
    $LE\_bowen\_corr = \text{NaN}$
    $H\_bowen\_corr = \text{NaN}$
  end if
  if $LE\_bowen\_corr = \text{NaN}$ or $H\_bowen\_corr = \text{NaN}$ **then**
    Del *df_site_daily*
  end if
  Del $LATENT\_SENSIBLE$ & $NETRAD\_SOIL$
  Go to **Algorithm [8]**
  $LE\_bowen\_corr(mm) = LE\_bowen\_corr \ / \ 28.94$
  $H\_bowen\_corr(mm) = H\_bowen\_corr \ / \ 28.94$
  Sort columns
  Export *df_site_daily* into **FLUX_DAILY_PATH**
  Add *df_site_daily* to sites
end for
Export sites to **FLUX_DAILY_PATH**

6.1.5 Reference Evaporation Calculation

This section focuses on computing ET_o and ET_r for all the data sets

1. Add ET_o to Daily Combined Sites

   (a) Load data sets from path **FLUX_PROCESSED_DAILY_PATH**

   (b) Compute the minimum, and maximum air temperatures for each date by achieving the following:

      i. Load the hourly data from path **FLUX_PATH**

ii. Re-sample the hourly data to daily by computing the minimum and maximum air temperature on the hourly data and summing them up and grouping the data by Site Id and Date fields

iii. Filter the sites prefixed with "US-" which indicates the Ameriflux sites

iv. Export the filtered sites into a sheet that holds the Ameriflux TA information to path **AMERIFLUX_TA**

v. Filter the sites not prefixed with "US-" which indicates the Euroflux sites

vi. Export the filtered sites into a sheet that holds 695874roflux TA information to path **EUROFLUX_TA**

(c) Inner merge each site's data with its respective TA information data set

(d) Compute the day of the year

(e) Compute ET_o as described in **Algorithm [2]**

(f) Compute the fraction reference evapotranspiration (ETrF_bowen) by dividing LE_bowen_corr(mm) by ET_o

2. Add ET_o to the daily combined sites

3. Repeat the same above procedure to compute ET_r by invoking ".etr()" method and supplying the same field values and adding the generated columns to the data set

4. Export each data set to path **FLUX_PROCESSED_DAILY_PATH** with the site name suffixed with "_daily" for instance: US-Var_daily.csv

5. Concatenate all data sets into one data set

6. Export the concatenated data set to path **FLUX_PROCESSED_DAILY_PATH** with the name "library_corrected_daily.csv"

6.1.6 EEflux Joining with Tower Data

This section focuses on merging the EEFlux data with the tower data to use for our study to compare tower versus EEflux data

1. Load the concatenated data set from path **FLUX_PROCESSED_DAILY_PATH**

2. Get a list of unique sites from the above sheet

3. Load the data set having EEFlux Evapotranspiration for all sites from path **EEFLUX_ET_PATH**

4. Loop over each site
5. Inner merge the loaded data set with the flux daily data set
6. If the current site has a prefix "US-" then do the below
   (a) Load EEflux ET_o from path **AMERIFLUX_ETo**
   (b) Merge the data set generated above with the one loaded from **AMERIFLUX_ETo**
   (c) Load EEflux ET_r from path **AMERIFLUX_ETr**
   (d) Merge the data set generated above with the one loaded from **AMERIFLUX_ETr**
   (e) Load EEflux Albedo and NDVI data set from path **EEFLUX_ALBEDO_NDVI**
7. If the current site does not have a prefix "US-" then do the below
   (a) Load EEflux ET_o from path **EUROFLUX_ETo**
   (b) Merge the data set generated above with the one loaded from **EUROFLUX_ETo**
   (c) Load EEflux ET_r from path **EUROFLUX_ETr**
   (d) Merge the data set generated above with the one loaded from **EUROFLUX_ETr**
   (e) Load EEflux Albedo and NDVI data set from path **EEFLUX_ALBEDO_NDVI**
8. Set EEflux ET initially to Modeled ET, otherwise, if it is NaN then set it to Mean Modeled ET
9. Set EEflux ETo initially to Modeled ETo, otherwise if it is NaN then set it to Mean ETo
10. Compute the fraction reference evapotranspiration (EEflux ETrF) by dividing EEflux ET by EEflux ET_r and then repeating the same procedure for the EEflux EToF
11. Drop all the previous generated lags for each site's data
12. Re-generate lags for the main variables
13. Order the columns following the daily ordering mentioned earlier
14. Drop NaN columns i.e columns having all their rows as NaNs meaning no valid value at all
15. Export each data set to path **FLUX_JOINT_PATH** with the site name suffixed with "_Joint.csv"

16. Concatenate all the data sets
17. Export the concatenated data sets to path **FLUX_JOINT_PATH** with the name "manual_corrected_joint.csv"

6.2 Library Data Generation

This section focuses on performing all the necessary steps to produce a daily corrected data set and another that is joint with EEflux using the library Bowen and EBR corrections. Steps involved are:

1. Global Variable Initialization
2. Configuration Generation
3. Data Cleaning
4. Data Correction
5. Data Correction Processing
6. Data Aggregation
7. Reference Evapotranspiration Calculation
8. EEflux Joining with Tower Data

6.2.1 Global Variables Initialization

- Set BASE_PATH: "Users/saraawad/Desktop/Datasets/Google/"
- Set INPUT_PATH: "/Users/saraawad/Hourly Cleaned/Ameriflux/"
- Set FLUX_PATH: "/Users/saraawad/Desktop/flux-data-qaqc/sites/data"
- Set FLUX_CONFIG_PATH: "/Users/saraawad/Desktop/flux-data-qaqc/sites/output/"
- Set FLUX_EBR_PATH: "/Users/saraawad/Desktop/flux-data-qaqc/sites/output/EBR/"
- Set FLUX_BOWEN_PATH:"/Users/saraawad/Desktop/flux-data-qaqc/sites/output/Bowen
- Set CORRECTION_METHOD = 1 for EBR or 2 for Bowens correction
- Set isEBR = true for EBR and false for Bowens
- Set FLUX_EBR_BOWEN_PROCESSED_PATH: BASE_PATH + "Common/Daily/"
- Set FLUX_COMBINED_PATH: BASE_PATH + "Common/Daily Combined/

- Set AMERIFLUX_TA: BASE_PATH + “Ameriflux_TA.csv”
- Set EUROFLUX_TA: BASE_PATH + “Euroflux_TA.csv”
- Set FLUX_DAILY_PATH: BASE_PATH + “Common/Ref/”
- Set EEFLUX_ET_PATH: BASE_PATH + “EEflux/EEflux_sites.csv”
- Set EUROFLUX_ETo_PATH: BASE_PATH + “Euroflux_ETo_grass.csv”
- Set EUROFLUX_ETr_PATH: BASE_PATH + “Euroflux_Potential_ET.csv”
- Set AMERIFLUX_ETo: BASE_PATH + “Ameriflux_ETo_grass.csv”
- Set AMERIFLUX_ETr: BASE_PATH + “Ameriflux_ETr_alfafa.csv”
- Set EEFLUX_ALBEDO_NDVI: BASE_PATH + “EEflux/EEflux_Albedo_NDV.csv”
- Set FLUX_JOINT_PATH: BASE_PATH + “Common/Joint Combined/

6.2.2 Configuration Generation

This section generates configuration file for each site

1. Copy Euroflux hourly data sets to **FLUX_PATH**
2. Check if all the main variables i.e (”NETRAD”, ”H”, ”LE”, ”G”, ”RH”, ”WS”, ”TA”) exists as columns
3. If they exist, then generate the configuration file which ends in “.ini” and contains meta data and variable information as described in **Chapter [3]**
4. Export the “.ini” files to path: **FLUX_CONFIG_PATH**

6.2.3 Data Cleaning

This section handles data cleaning the same way it is handled for the manual data generation correction

6.2.4 Data Correction

This section applies Bowen and EBR Correction methods by internally converting the hourly into daily data sets

1. Load the configuration file for each site from **FLUX_CONFIG_PATH**
2. Start with the EBR correction by setting the **CORRECTION_METHOD = 1**

3. The algorithm will run as described in **Chapter** [**5**] and exports a daily and a monthly version for each site according to the predefined configuration file, therefore, yielding the new data sets to **FLUX_PATH**

4. Delete the monthly data sets since they are not needed for our study

5. Move the exported files to **FLUX_EBR_PATH**

6. After having finished from the EBR Correction, repeat the same steps and set the **CORRECTION_METHOD** = 2 to perform the Bowen's correction and move the generated files to the path: **FLUX_BOWEN_PATH**

Algorithm 5 Library Cleaning and Correction

Data:

Load hourly data from **INPUT_PATH** into *df_hourly*

Del columns with suffix "_SSITC_TEST" & "WS_MAX"
Del NaN rows
Del columns with suffix "_1", "_2", "_3"
Export *df_hourly_cl* into **FLUX_PATH**
Load hourly data from **FLUX_PATH** into *df_hourly_cl*
for *site_df in df_hourly_cl* **do**
    Set $MAIN\_VARS$ = ["NETRAD", "H", "LE", "G", "RH", "WS", "TA"]
    if $MAIN\_VARS$ exists **then**
        $Site\_id\_config \leftarrow$ Generate config for *site_df*
    end if
    Export $Site\_id\_config$ into **FLUX_CONFIG_PATH**
end for
for *site_df in df_hourly_cl* **do**
    Load config data from **FLUX_CONFIG_PATH** into df_config
    $CORRECTION\_METHOD = 1$
    site_df_corrected $\leftarrow$ EBRCorrection(*site_df*) & df_config
    Del month data
    Export *site_df_corrected* into **FLUX_EBR_PATH**
end for
for *site_df in df_hourly_cl* **do**
    Load config data from **FLUX_CONFIG_PATH** into df_config
    $CORRECTION\_METHOD = 2$
    site_df_corrected $\leftarrow$ BowenCorrection(*site_df*) & df_config
    Del month data
    Export *site_df_corrected* into **FLUX_BOWEN_PATH**
end for

6.2.5 Data Correction Processing

This section focuses on cleaning and processing each library correction generated data set

1. Loop over all the unique sites from sheet "filtered_sites_encoded_all.xlsx"
2. Load the data set from path **FLUX_EBR_PATH**
3. Set **isEBR** to true
4. Perform a binary encoding to the Site Id, Vegetation, Climate, and Month columns as mentioned in **Chapter [5]**

5. Refactor the generated corrected columns so that to differentiate the ones coming from EBR or Bowens method by adding a suffix i.e "_ebr" or "_bowen" to the main columns

6. Generate date components fields i.e Year, Month, Day columns

7. Check if the main columns exist i.e ("NETRAD", "H", "LE", "G","RH","WS", "TA") e, if so continue, otherwise exit and do not process the site data further

8. Generate lags for the columns: "H", "H_ebr_corr", "H_bowen_corr", "G", "NETRAD", "WS", "RH", "TA"

9. Calibrate LE, ET and H by adding a conversion rate to each of "LE_ebr_corr", "LE_bowen_corr", "ET_ebr_corr", "ET_bowen_corr", "H_ebr_corr" and"H_bowen_corr" i.e dividing each by 28.94. The resulting columns will have a suffix (mm)

10. Order columns following the below criteria

    (a) Columns coming from the filtered sheet i.e Date, Site Id, Year, Month, Day, Climate, Vegetation, Latitude, Longitude, Elevation(m)

    (b) Weather columns i.e WS, RH, TA, NETRAD, SW_IN followed by their lags i.e WS-1, WS-2,.. WS-5, RH-1, RH-2,.. RH-5, etc..

    (c) Energy balanced columns i.e G, H, H_ebr_corr, H_ebr_corr(mm), H_bowen_corr, H_bowen_corr(mm), LE, LE_ebr_corr, LE_ebr_corr(mm), LE_bowen_corr, LE_bowen_corr(mm), ET_ebr, ET_ebr_corr, ET_ebr_corr(mm), ET_bowen_corr, ET_bowen_corr(mm)

    (d) Reference ET columns i.e ETo, ETr, ETof_ebr, ETof_bowen, ETrf_ebr, ETrf_bowen

    (e) EEflux columns if exists i.e EEflux Albedo, EEflux NDVI, EEflux LST, EEflux ET, Cloud Cover, EEflux ETo, EEflux ETr

11. Export the processed EBR data set for each site to path **FLUX_EBR_BOWEN_PROCESSED_PATH** with the site id suffixed with "ebr_daily.csv" for instance: US-Var_ebr_daily.csv

12. Repeat the same procedure for the Bowens data by loading the data from **FLUX_Bowen_PATH** and setting **isEBR** to false

13. Export the processed Bowen data set for each site to path **FLUX_EBR_BOWEN_PROCESSED_PATH** with the site id suffixed with "bowen_daily.csv" for instance: US-Var_bowen_daily.csv

Algorithm 6 Library Correction Processing

Data:

Load EBR corrected daily data from **FLUX_EBR_PATH** into *df_ebr*
Load Bowen daily corrected data from **FLUX_BOWEN_PATH** into *df_bowen*
Load sites data from sheet **filtered_sites_encoded_all** into *sites_df*

Get unique sites from *df_hourly*
for *site in unique_sites* **do**
  Binary encode "Site_Id", "Vegetation", "Climate", & "Month"
  $CORRECTED\_COLUMNS$ = ["*LE_corr*", "*H_corr*", "*ET_corr*"]
  for col in $CORRECTED\_COLUMNS$ **do**
    if *df_ebr* **then**
      col ← col + "_ebr"
    else if *df_bowen* **then**
      col ← col + "_bowen"
    end if
  end for
  Compute "Year", "Month", & "Day" from "Date" column
  Set $MAIN\_VARS$ = ["*NETRAD*", "*H*", "*LE*", "*G*", "*RH*", "*WS*", "*TA*"]
  if $MAIN\_VARS$ not exists **then**
    continue
  end if
  Go to **Algorithm [8]**
  Set $CALIBRATE\_VARS$ = ["LE_ebr_corr","LE_bowen_corr", "ET_ebr_corr", "ET_bowen_corr", "H_ebr_corr", "H_ebr_corr"]
  for *col in* $CALIBRATE\_VARS$ **do**
    $LE\_bowen\_corr(mm)$ = *LE_bowen_corr* / 28.94
  end for
  Sort columns
  Export *df_ebr* into **FLUX_EBR_PROCESSED_DAILY_PATH**
  Export *df_bowen* into **FLUX_EBR_PROCESSED_DAILY_PATH**
end for

6.2.6 Data Aggregation

This section focuses on merging the EBR and Bowen correction data sets into one excel for each site

1. Get a list of unique sites from sheet "filtered_sites_encoded_all.xlsx"

2. Loop over each site and load the EBR data set into one data set and its respective Bowen into another data set

3. Perform an outer merge on both data sets on columns “Site Id” and “Date” which are unique per row

4. Export each data set to path **FLUX_COMBINED_PATH** with the site name suffixed with “_corrected_daily” for instance: US-Var_corrected_daily.csv

Algorithm 7 Library Correction Aggregation

Data:

Load corrected data from **FLUX_EBR_PROCESSED_DAILY_PATH** into *df_daily*
Load sites data from sheet **filtered_sites_encoded_all** into *sites_df*

```
Get unique sites from sites_df
for df_site in unique_sites do
    for df_site in df_daily do
        FILE_NAME_SUFFIX ← df_site name
        if FILE_NAME_SUFFIX ends “ebr” then
            Add df_site to df_ebr
        else
            Add df_site to df_bowen
        end if
        Outer Merge df_ebr & df_bowen on SiteId & Date into df_daily_agg
        Export df_daily_agg to FLUX_COMBINED_PATH
    end for
end for
```

6.2.7 Reference Evaporation Calculation

This section focuses on computing ET_o and ET_r for all the data sets as described in the section for the manual data correction generation, it only differs in the path we import the data from i.e **FLUX_COMBINED_PATH** and to the path we export the final data set to i.e **FLUX_DAILY_PATH** with the name “library_corrected_daily.csv”

6.2.8 EEflux Joining with Tower Data

This section focuses on merging the EEFlux data with the tower data as described in the section for the manual data correction generation, it only differs in the path we import the data from i.e **FLUX_DAILY_PATH**

6.3 Data Description

The final generated data sets are six data sets as such:

6.3.1 Data sets

1. Manual Bowen Daily Data set: This is a daily data set obtained by applying the manual Bowen correction method on our daily data set. It contains all the below fields excluding columns with the suffix "ebr" and EEflux columns other than Albedo, NDVI & LST.

2. Library Bowen Daily Data set: This is a daily data set obtained by applying the library Bowen correction method on our daily data set. It contains all the below fields excluding columns with the suffix "ebr" and EEflux columns other than Albedo, NDVI & LST.

3. Library Ebr Daily Data set: This is a daily data set obtained by applying the library EBR correction method on our daily data set. It contains all the below fields excluding columns with the suffix "bowen" and EEflux columns other than Albedo, NDVI & LST.

4. Manual Bowen Joint Data set: This is a daily data merged with EEflux data set obtained by applying the manual Bowen correction method on our daily data set and merging it with EEflux fields. It contains all the below fields excluding columns with the suffix "ebr".

5. Library Bowen Daily Data set: This is a daily data merged with EEflux data set obtained by applying the library Bowen correction method on our daily data set and merging it with EEflux fields. It contains all the below fields excluding columns with the suffix "ebr".

6. Library Ebr Daily Data set: This is a daily data merged with EEflux data set obtained by applying the library Ebr correction method on our daily data set and merging it with EEflux fields. It contains all the below fields excluding columns with the suffix "bowen".

6.3.2 Data Legend

Each data set consists of all or some of the below fields:

- *Date*: The daily date at which the data was recorded at with the format "MM/dd/YY"

- *SiteId*: A unique identifier for each site

- *Year*: The year at which the data was recorded

- *Month*: The month at which the data was recorded

- *Day*: The day at which the data was recorded

- *Elevation*: The elevation in meters
- *Latitude*: The latitude
- *Longitude*: The longitude
- *Climate*: If available one of 5 values (Cfa, Csa, Csb, Dsa, Cwa) or Other for sites coming from Euroflux
- $Vegetation(IGBP)$: Values are one of which (CRO, GRA, ENF, CSH, EBF, DBF, WET, WSA)
- WS: Wind speed
- RH: Relative humidity
- TA: Air temperature
- $NETRAD$: Net radiation
- SW_IN: Incoming Shortwave Solar Radiation
- G: Ground/Soil flux
- H: Sensible heat flux
- H_ebr_corr: The corrected sensible heat flux using EBR method
- H_bowen_corr: The corrected sensible heat flux using Bowen method
- LE: Latent energy flux
- LE_ebr_corr: The corrected Latent energy flux using EBR method
- $LE\_ebr\_corr(mm)$: The calibrated LE_ebr_corr
- LE_bowen_corr: The corrected Latent energy flux using EBR method
- $LE\_bowen\_corr(mm)$: The calibrated LE_bowen_corr
- ET_ebr: The evaportanspiration calculated from LE and TA using EBR method
- ET_ebr_corr: The corrected evaportanspiration calculated from LE and TA using EBR method
- $ET\_ebr\_corr(mm)$: The calibrated ET_ebr_corr
- ET_bowen: The evaportanspiration calculated from LE and TA using Bowen method

- ET_bowen_corr: The corrected evaportanspiration calculated from LE and TA using Bowen method
- $ET\_bowen\_corr(mm)$: The calibrated ET_bowen_corr
- $gridMET\_ETr$: The gridMET alfalfa reference ET (nearest cell)
- ET_r: The alfafa reference evaportanspiration
- ET_{rF}_bowen: The fraction reference evaportanspiration for ET_bowen_corr, i.e.ET_bowen_corr /ET_r
- ET_{rF}_ebr: The fraction reference evaportanspiration for ET_ebr_corr, i.e.ET_ebr_corr / ET_r
- ET_o: The grass reference evaportanspiration
- ET_{oF}_bowen: The fraction reference evaportanspiration for ET_bowen_corr, i.e.ET_bowen_corr / ET_o
- ET_{oF}_ebr: The fraction reference evaportanspiration for ET_ebr_corr, i.e.ET_ebr_corr / ET_o
- $EEfluxET$: ET from EEflux which initially has the value of ET, if ET is empty it is populated from the mean of the 8 neighboring pixels
- $EEfluxET_o$: The grass reference from the Climate Engine
- $EEfluxET_r$: The grass reference from the Climate Engine
- $EEfluxLST$: Land Surface temperature from EEflux
- $EEfluxNDVI$: NDVI from EEflux
- $EEfluxAlbedo$: Albedo from EEflux
- $column\_number$: The site id, month, and vegetation columns are encoded using binary encoding with a format of the column name followed by the number of the encoding column, i.e $SiteId\_1$
- $column - number$: This is a lag of the column name i.e $RH - 1$ is the first lag of RH, $RH - 2$ is the second lag of RH

We thus denote the LE corrected by Real ET.

6.3.3 Daily-Joint Mapping

Having two types of data sets. i.e daily and joint and as has been mentioned above, the daily data set constitutes daily instances, and the joint data set constitutes weekly instances. The joint data set contains EEflux ET that is not available on our daily data set, we will further need to create a mapping between the daily and joint data sets as follows:

1. Split the data set into a training and testing data set
2. Save the testing data set
3. Perform modeling
4. Add the predicted ET to the testing data set and refer to this data set to be the daily data set
5. Merge the daily data set with the weekly data set (the data set having EEflux ET) on the unique fields
    - Site Id
    - Date
    - Real ET
6. Export the final data set which contains the real ET predicted ET, and EEflux ET that is a subset of the daily data set that is mapped to the joint data set.

6.3.4 Choice of Data set

To decide on which data sets we should be using, we have performed several experimental evaluations and assessed the performance of each as such:

- Compare Bowen versus EBR Correction: Models trained on the Library Bowen daily data set yielded better results than those trained on the Library EBR daily data set by an increase of 77.37% in the model's accuracy.
- Compare Library Bowen Correction versus Manual Bowen: Models trained on the Manual Bowen daily data set yielded better results than those trained on the Library Bowen daily data set by an increase of 4.8% in the model's accuracy.

Thus, our chosen data set is the Manual Bowen Daily data set which was the best amongst them all.

6.3.5 Final Data set

Each final data set is divided into input features and an output variable as such. The Manual Data set consists of 10,911 rows.

- Input: Lags are generated for each of the following input features as been defined by auto-correlation plots by adding a suffix of "-lag number". For instance "RH-1" represents the first lag.
  - Wind speed (WS)
  - Air temperature (TA)
  - Relative humidity (RH)
  - Net radiation (EEflux NDVI)
  - EEflux Albedo
  - Land Surface Temperature (EEflux LST)
  - Vegetation Encoded: The vegetation is encoded using binary encoding with a format of the column name followed by the number of the encoding column, i.e Vegetation_1, Vegeation_2, etc...
  - Site ID Encoded: The site ID is encoded using the same strategy as the vegetation.
  - Month Encoded: The month is encoded using the same strategy as the vegetation.
  - EEflux ET: Modeled ET coming from EEflux.
- Output: real ET: The calibrated and corrected real ET using Bowen's method.

Chapter 7

Feature Engineering

The feature engineering procedure involves studying the distribution of our features, performing exploratory analysis techniques, and applying different data transformations.

7.1 Time-series Analysis

Our problem is a time-series problem since it exhibits the data component. We will perform time-series analysis to ensure our data is stationary and further can be modeled.

7.1.1 Stationary Study

A time series is stationary if it does not exhibit any trend or seasonal effect. Summary statistics should be consistent among observations for a problem to be predictable. Several approaches exist for checking if a time series is stationary or not, some of which are as follows:

- Summary statistics i.e (mean, variance) should be consistent over time
- Statistical tests like Augmented Dickey-Fuller that identifies if the data is stationary or not based on hypothesis tests.

Augmented Dickey-Fuller (ADF) test is a statistical test known as a unit root test. The instinct behind a unit root test is that it decides how firmly a time series is characterized by a trend. It is used to identify if a time series is stationary or not (Brownlee, 2016). The following are the hypotheses of the test:

- Null Hypothesis (H0): If failed to be rejected, it means that the time series has a unit root, thus it is non-stationary. It has some time-dependent structure.

- Alternate Hypothesis (H1): The null hypothesis is rejected; it means that the time series does not have a unit root, thus it is stationary. It does not have time-dependent structure.

We have applied ADF test to our data which yielded the results in **Figure** [**7.1**].

```
ADF Statistic: -15.047017
p-value: 0.000000
Lags used: 23
Critical Values:
1%: -3.960
5%: -3.411
10%: -3.127
-----------------------------------------------
```

Figure 7.1: ADF Results

Figure [**7.1**] shows that the test statistics i.e (-15.04) is lower than all the critical values and the p-value ≤ 0.05 meaning we reject the null hypothesis (H0) and thus the data is stationary.

7.1.2 White-Noise Study

White noise is considered an essential concept in time series prediction. A time series is a white noise if its variables are identically distributed and independent with a mean of zero. If a series is a white noise then it resembles a sequence of random numbers and thus cannot be predicted.

ljung_box_test is used to test if a white noise exists or not. The following are the hypotheses of the test:

- H0: The data are independently distributed (i.e. the correlations in the population from which the sample is taken are 0 so that any observed correlations in the data result from the randomness of the sampling process)
- H1: The data are not independently distributed; they exhibit serial correlation

Conditions to accept/refute the null hypothesis:

- p-value <= 0.05: reject the null hypothesis in favor for the alternative hypothesis
- p-value > 0.05: fail to reject the null hypothesis.

Given that the test in **Figure** [**7.2**] shows that all p-values are less than 0.05, the time series doesn't exhibit white noise.

```
all p-values for ljung box <= 0.05
all p-values for box pierce <= 0.05
-------------------------------------------
```

Figure 7.2: ljung Box Results

7.1.3 Random Walk Study

A time series is a random walk if one of the following is observed:

- The time series shows a strong temporal dependence that decays linearly or in a similar pattern.
- The time series is non-stationary and making it stationary shows no learnable structure in the data.

We generate auto-correlation plots to study the relationship between lags. We do that for each input and output feature in **Section [7.3.1]**.

Given the above, the time series doesn't exhibit a random walk since auto-correlation plots i.e (for TA, RH, etc..) show a significant relationship between the lagged observations.

7.2 Exploratory Data Analysis

In this section, we will analyze our data set by studying the distribution of each feature, generating some summary statistics, and visualizing some plots. We use the following tools to generate our interpretations:

- pandas-profiling
- facets
- fitter
- matplotlib

7.2.1 Summary Statistics

In this section, the tools used to generate the below are facets and pandas-profiling.

We will first start with inspecting the number of rows in our data set.
As shown in **Table [7.2.1]**, the number of records in the data set is 10,916.

Number of Observations	10916
Total Missing (%)	0.00%
Total size in memory	4.3 MiB
Average record size in memory	416.0 B

7.2.2 Data Sources Distribution

In this section, the tool used to generate the below is matplotlib.

We will then visualize how many sites there are for Ameriflux and Euroflux.

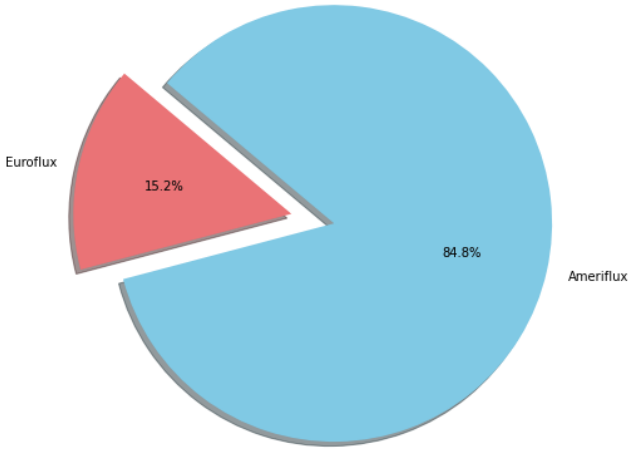


Figure 7.3: Data Sources of Sites

As per **Figure [7.3]**, we note that 84.8% of the sites come from Ameriflux and only 15.2% come from Euroflux.

We will then visualize the number of rows for each site

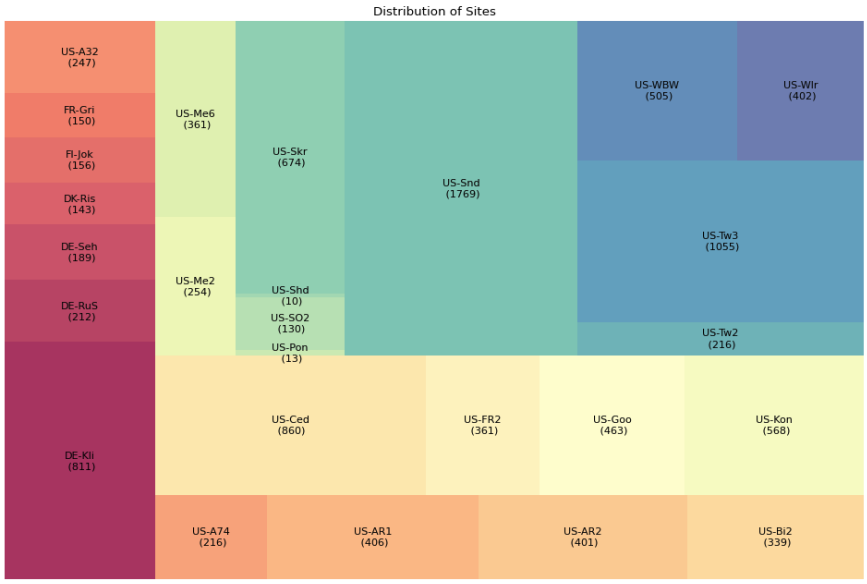


Figure 7.4: Distribution of Sites

As per **Figure** [**7.4**], the top five representative sites (sites having the highest number of rows) are as such:

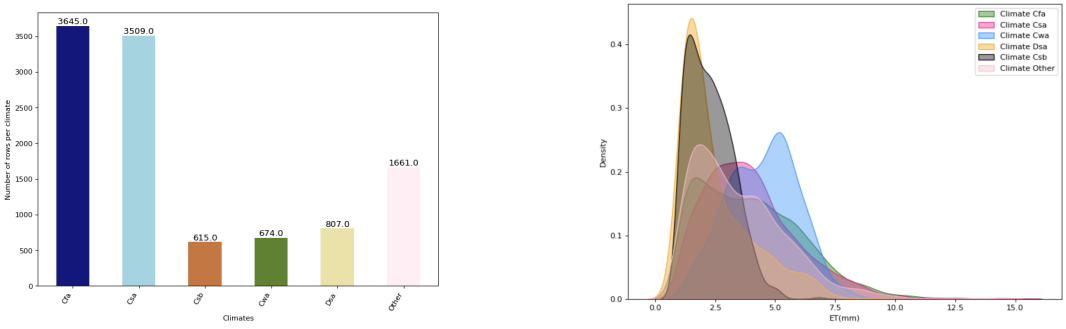
- US-Snd: This is a site pertaining to Ameriflux. It consists of 1,769 rows.
- US-Tw3: This is a site pertaining to Ameriflux. It consists of 1,055 rows.
- US-Ced: This is a site pertaining to Ameriflux. It consists of 860 rows.
- DE-Kli: This is a site pertaining to Euroflux. It consists of 811 rows.
- US-Skr: This is a site pertaining to Ameriflux. It consists of 674 rows.

7.2.3 Climate Distribution

In this section, the tool used to generate the below is matplotlib.

We will then plot the distribution of all the climates.
As per **Figure** [**7.5a**], we note that the highest number of rows are for climates Cfa and Csa (almost 3,000 rows). Climates Cwa, Dsa, and Csb have a comparable number of rows (in the hundreds).

Figure [**7.5**] shows the density plots for the target variable for each climate. We note that climates Dsa and Csb show a similar distribution of ET values ranging from 0 - 5 mm and no rare values are included. However, climates (Csa, Cfa, Cwa, and Other) somehow show a similar distribution of ET values ranging



(a) Climates Histogram

(b) Climates Density Plots

Figure 7.5: Distribution of Climates

from 0 - 12.5 mm, but having rare values.

As been previously noted in **Figure [7.4]**, the top five sites with their respective climates are as follows:

- US-Snd and US-Tw3: These sites have a Csa climate (Mediterranean: mild with dry, hot summer).
- US-Ced: This site has a Cfa climate (Humid Subtropical: mild with no dry season, hot summer).
- DE-Kli: This site has an unknown climate labeled as Other, since Euroflux does not indicate the climate of each site, we further labeled them as Other.
- US-Skr: This site has a Cwa Climate (Humid Subtropical: dry winter, hot summer)

Thus, we note that the top three sites are represented by the top two climates (Csa and Cfa).

7.2.4 Seasonal Distribution

In this section, the tool used to generate the below is matplotlib.

Figure [7.6] shows the density distribution of the target variable ET for the spring, summer, and winter seasons. We note that the summer and spring seasons include rare values as they have ET values greater than 5 mm unlike the winter season. Both seasons show comparable density distributions, however, the winter season shows a different density distribution with a peak value at (2.5 mm).

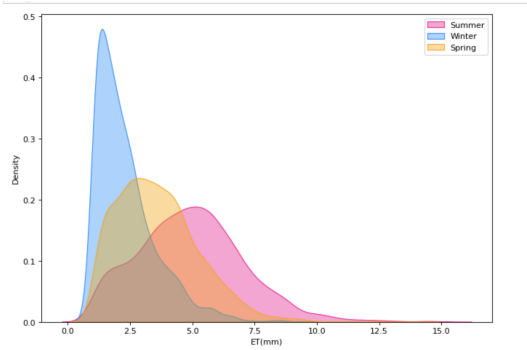
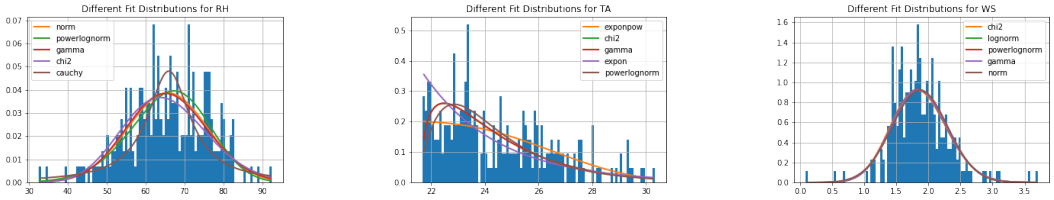


Figure 7.6: Seasons Distribution

7.2.5 Distribution of Input Features

In this section, the tool used to generate the below is fitter.

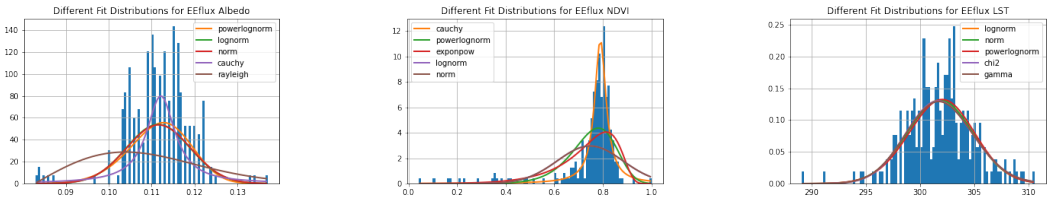
We will further check the distribution for each of our input features on one of the most representative sites in our data: **US-Ced**. We use a package that helps in identifying the underlying data distribution by comparing the histogram of the given data with a probability density function of a known distribution i.e normal. This package is available under fitter. It consists of 80 different distributions from Scipy and also a list of the common ones to try out on the data, some of which are as such: "chi2", "exponential", "exponpower", "uniform", "log", etc... It returns the best fit according to the distribution having the least sum of squares error. It also supports plotting the results to check which distributions happen to be the best.



(a) RH (b) TA (c) WS

Figure 7.7: Distributions of RH, TA & WS

- The best fit for RH is "norm"
- The best fit for TA is "exponpow"
- The best fit for WS is "chi2"



(a) Albedo (b) NDVI (c) LST

Figure 7.8: Distributions of EEflux Albedo, NDVI & LST

- The best fit for Albedo is "powerlognorm"
- The best fit for NDVI is "cauchy"
- The best fit for LST is "lognorm"

Since all our input features do not have the same distribution, we will use a min-max scaler to scale all the input features with a scale between 0 and 1.

7.2.6 Distribution of Output Variable

In this section, the tool used to generate the below is fitter.
The best fit for ET is "powerlognorm".

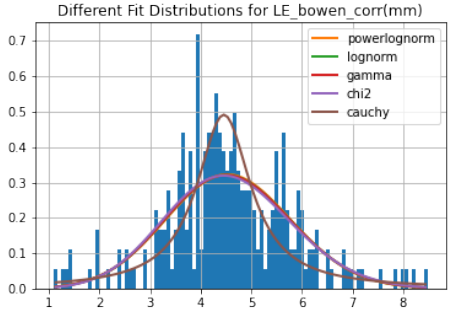


Figure 7.9: Distribution of LE

7.2.7 Seasonal Trends of TA and RH alongside ET

In this section, the tool used to generate the below is matplotlib
In **Figure [7.10]**, in the first row, the x-axis represents TA in degrees Celsius versus ET in mm and the y-axis represents the number of data points.
In the second row, the x-axis represents RH in (%) versus ET in mm and the y-axis represents the number of data points. The first column shows the data for

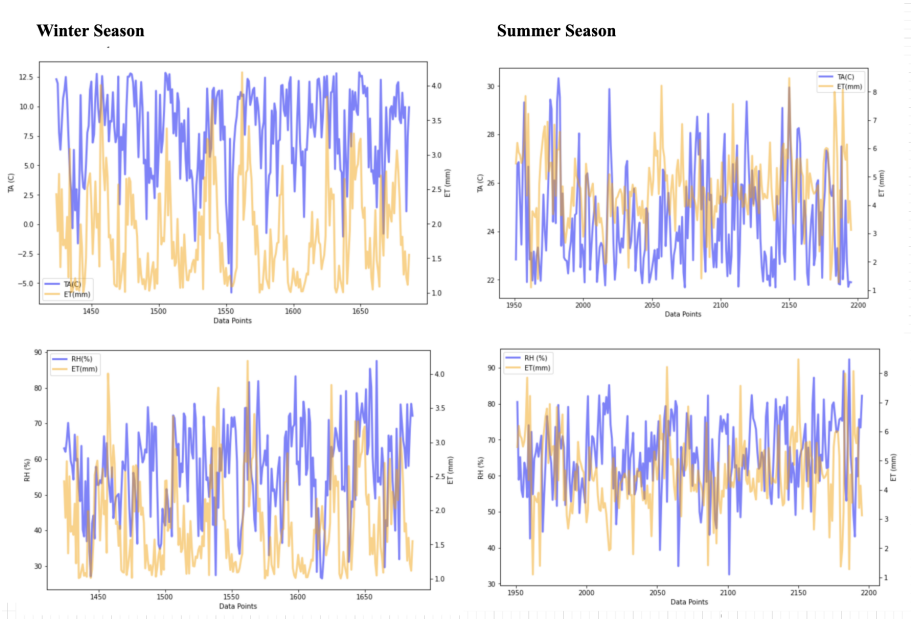


Figure 7.10: Ameriflux Site: US-Ced

the winter season and the second columns shows the data for the summer season.

We are studying how TA, RH relate to ET as the seasons change. As shown in **Figure [7.10]**, TA and ET are directly proportional in the summer (high TA implies high ET) and winter (Low TA implies low ET) seasons. This aligns with the observations in **Chapter [9]** and **Chapter [16]**.

RH and ET are inversely proportional in the summer (low RH implies high ET) and winter (high RH implies low ET) seasons. This also aligns with the observations in **Chapter [9]** and **Chapter [16]**.

We now observe another top representing site from Euroflux (DE-Kli) We note that in **Figure [7.11]**, the same observations were shown as in **Figure [7.10]**.

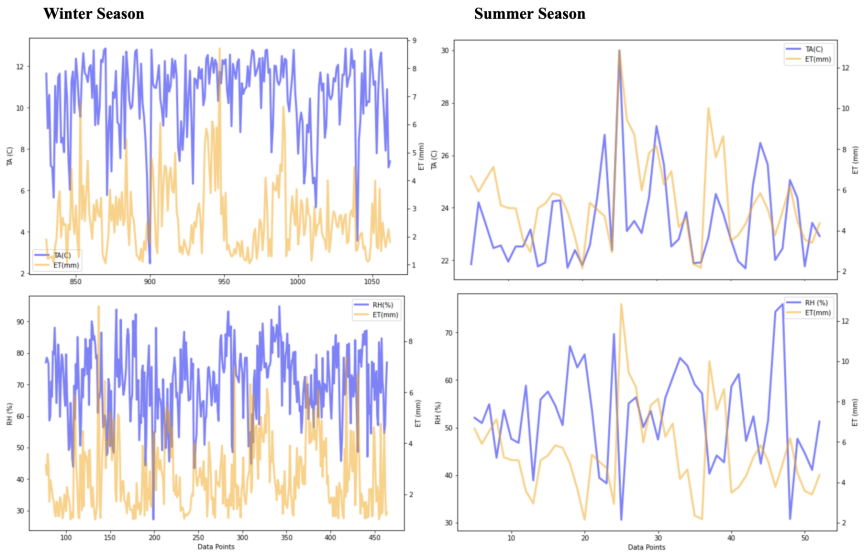


Figure 7.11: Euroflux Site: DE-Kli

7.3 Data Transformations

Several data transformations will be applied to our data set. Since our data set contains a date component representing a temporal structure, we will create lags for a set of input columns so that we get rid of the date column in our modeling and use the lags to symbolize the temporal structure. To create lags, we will need to perform auto-correlation plots for each input feature. Moreover, we also binary encode our categorical columns since our models do not work well with categorical variables.

7.3.1 Autocorrelation

Correlation is a methodology that is used to quantify the strength and the type of relationship between an observation and its lag(s), this is also called auto-correlation. A correlation value ranges between -1 and 1. A value close to zero indicates a weak correlation, whereas a value closer to -1 or 1 indicates a strong negative or a strong positive correlation respectively.
We will be studying the correlation between an observation and its lags to select the best number of lags for each input feature. This study is performed using a variety of plots.

Lag Scatter Plots

Time series modeling assumes a relationship between an observation and the previous observation. Previous observations in a time series are called lags, with

the observation at the previous time step called lag=1, the observation at two-time steps ago lag=2, and so on. A useful type of plot to explore the relationship between each observation and a lag of that observation is called the scatter plot. Pandas have a built-in function for exactly this called the lag plot.
It plots the observation at time t on the x-axis and the lag=1 observation (t-1) on the y-axis.

- If the points cluster along a diagonal line from the bottom-left to the top-right of the plot, it suggests a positive correlation relationship
- If the points cluster along a diagonal line from the top-left to the bottom-right, it suggests a negative correlation relationship

We will plot each input feature as follows:

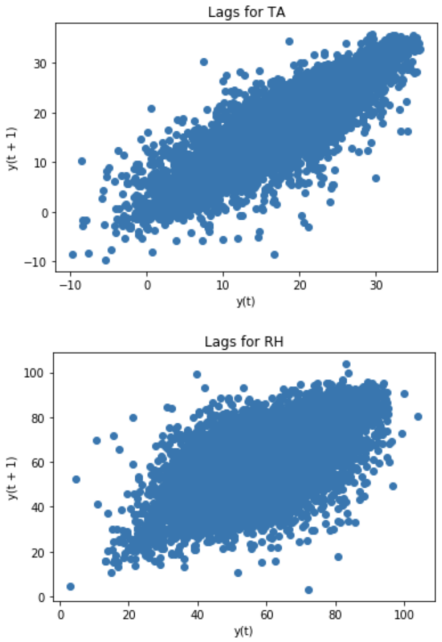


Figure 7.12: Scatter Plots for TA and RH

According to the above scatter plots, we notice that we have a positive correlation for the following features, as the scatter plot starts from the bottom left reaching the top right forming a diagonal shape which is an indication of a strong positive correlation:

- TA and its first lags
- EEflux LST and its first lags
- RH and its first lags

Other input fields do not show such an obvious pattern.

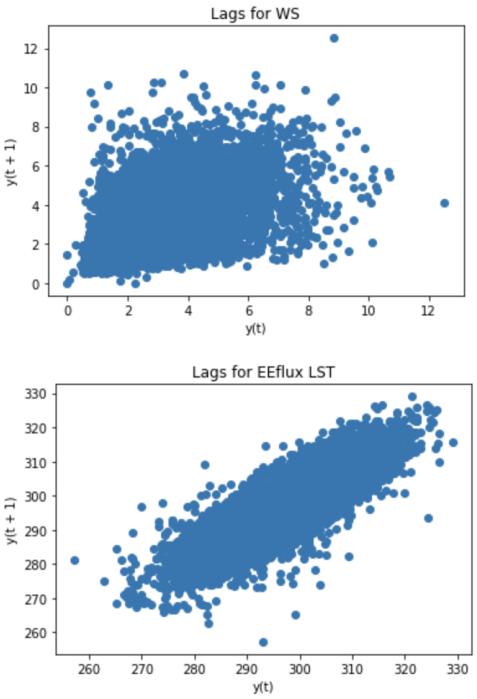


Figure 7.13: Scatter Plots for WS and LST

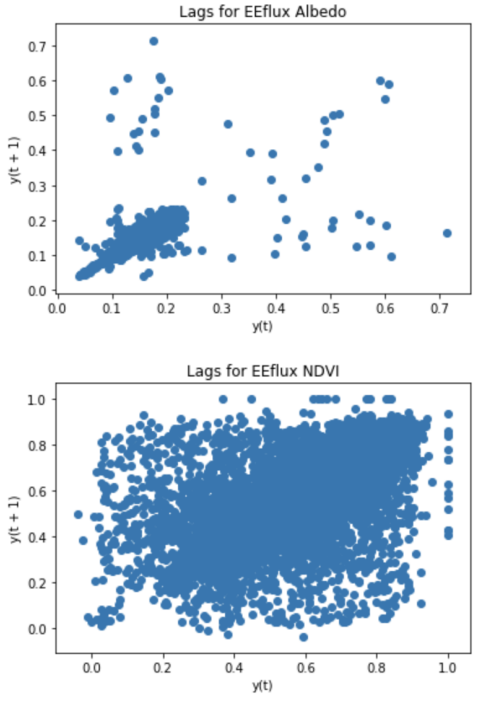


Figure 7.14: Scatter Plots for Albedo and NDVI

Autocorrelation Plots

We also represent visually this correlation using autocorrelation plots to better understand the relationship between an observation and its lag. The stronger the correlation between the output variable and a specific lagged variable, the more weight that autoregression model can put on that variable when modeling.
The most top features showing high and a positive correlation with their lags are TA, EEflux LST, and RH.

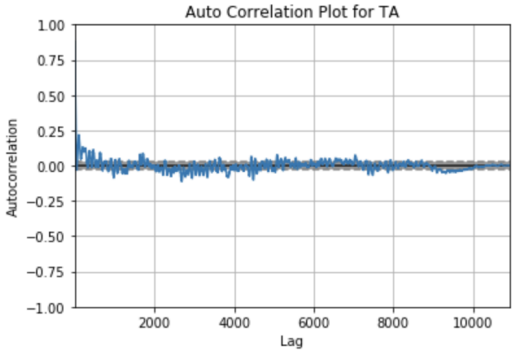


Figure 7.15: Autocorrelation Plot

Figure [7.15] shows on the x-axis the number of lags varying from no lags up to 1000+ lags and on the y-axis, it shows the auto-correlation as a value ranging from -1 to 1 representing a negative correlation and a positive correlation respectively. It shows the autocorrelation plot of TA and we note that increasing the numbers of lags to a certain extent will decrease the correlation value. The highest correlation value was 0.25 which aided in choosing the number of lags for TA to be 5. We repeat this figure for all our input features and we note that the best number of lags for each input feature is as such:

- TA: 5 lags
- EEflux LST: 5 lags
- RH: 3 lags
- WS: 2 lags
- EEflux Albedo: 2 lags
- EEflux NDVI: 2 lags

7.3.2 Lag Generation

We will generate lags for each input feature by varying $LAG\_FOR\_COLUMNS$ as per the number of lags required for each input feature by using the autocorrelation analysis above.

Algorithm 8 Lag Generation Algorithm

Data: Load data set into *df*
$LAG\_FOR\_COLUMNS$ = [“H_ebr_corr”, “H_bowen_corr”, “H”, “G”, “NETRAD”, “WS”, “RH”, “TA”, “SW_IN”, “EEflux Albedo”, “EEflux NDVI”, “EEFlux LST”]

```
Set LAGS_COUNT = 5
for col in LAG_FOR_COLUMNS do
    if col in df.columns then
        for lag_num in LAGS_COUNT do
            lag_name = col + "-" + lag_num
            df[lag_name] ← shift(df[col]) by lag_num
        end for
    end if
    return df
end for
return df
```

The above algorithm will yield the following output: “H-1”, ‘H-2”,.., “H-5” representing the 5 lags of “H”.

7.3.3 Data Encoding

We will be converting our categorical columns such as “Site Id”, “Month”, “Vegetation” and “Climate” into a numeric binary representation using a binary encoder. This technique works as such:

1. Each category is encoded as an ordinal representation.
2. Each ordinal/integer is converted into binary representation.
3. Each binary digit is split into a separate column with the value 0 or 1.

The reason for using this encoding is that it will result in fewer dimensions than a regular one-hot encoding which results in having a number of columns equal to the number of unique categories per column. Hence, this will increase our input feature dimensions drastically.

We use a binary encoding library to achieve that, it is installed by issuing:

```
pip install category_encoders
```

This will result in having variants for each column as such: “Site Id”, “Site Id_0”, “Site Id_1”, etc.. The number of columns produced for each will vary depending on the unique number of available categories per column.

7.4 Unsupervised Clustering Analysis

Several clustering algorithms are performed to study different patterns and visualizations of the output feature in our regression data set(s). Two main approaches are done: either data is being clustered automatically by using a clustering algorithm (Kmeans, Dendrograms) or by subsetting the data based on a range or a category for a set of input features.

7.4.1 Clustering by Kmeans and Dendrograms

KMeans

An unsupervised machine learning clustering algorithm that works by initially selecting random cluster centers (centroids) and then assigning each data point to its closest cluster by computing its distance from each centroid. The centroid will be updated by then computing the average of the assigned points and the procedure will be repeated until no further cluster update is done. This algorithm requires to specify the number of clusters, to do so for our problem the Elbow Method is used. Elbow method is a method that helps in selecting the best number of clusters which depends on computing, Within cluster Sum of Squares (WCSS), it is defined as the sum of the squared distance between each member of the cluster and its centroid.

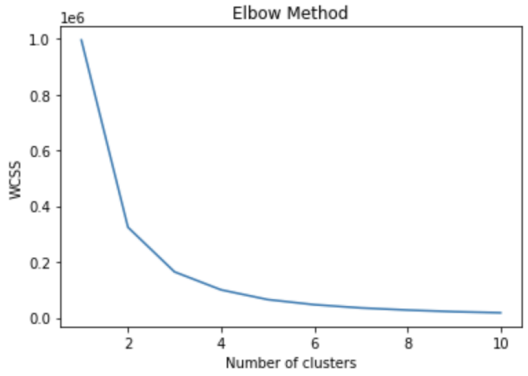


Figure 7.16: Elbow Method

In this figure, the elbow method shows the best k is when k=2, and k=3 so we will experiment with both.

Agglomerative Hierarchical Clustering

Is a clustering algorithm referred to as Dendrograms which works by assigning each point to an individual cluster and then merging the closest pair of clusters and repeating this step until only a single cluster is left? This merging procedure depends on calculating the similarity between data points by computing the distance between the centroids of these clusters in which the points having the least distance are referred to as similar points and thus we can merge them.

We perform several clustering experiments by varying the input features we fed to our algorithm.

- Clustering on one weather column only at a time (WS, RH, TA) and we perform that for $k = 2$ and $k = 3$.
- Clustering on two columns (WS RH, WS TA, RH TA) and we perform that for $k = 2$ and $k = 3$.
- Clustering on all three columns (WS, RH, TA) and we perform that for $k = 2$ and $k = 3$.
- Clustering on different climates encoded columns for $k = 4$.

The choice of k was based on running the elbow method on KMeans, finding the best cut, and trying with different k values. The aim of clustering by Kmeans or Dendrograms is to identify whether certain WS, TA, and RH values would yield better real ET predictions, rather than modeling on the whole data set with all ranges of WS, TA, and RH values.

7.4.2 Clustering by Subsetting on Climates

We divided our data into six different data sets (Cfa, Csa, Cwa, Dsa, Csb, and Other) by subsetting on the climate column and then model on each data set and experiment with when we have only one range or one climate at a time. This study is performed to indicate if we'd gain in performance when modeling on a certain climate versus when modeling on a union of climates.

7.5 Seasonality Study

After performing several clustering techniques, we are concerned with identifying if our data is separable by air temperature on a certain cluster/season. We thus perform a statistical and modeling assessment on clustering by air temperature (TA) using Kmeans with k=3. We have tried both Kmeans and Dendrograms, however, upon clustering with Dendrograms the data did not show a clear separation of seasons.

For our choice of k we have used the elbow method as shown in **Figure [7.16]**, where it shows that the best k is $k = 2$ and $k = 3$. We have tested on k=2, however, no clear separation of seasons was observed. Thus we have chosen k=3 as it clearly shows one season per cluster.

We note that clustering by TA has shown a clear clustering separability portraying seasonal representation of each cluster.

As shown in **Figure [7.17]**, it is noted that each figure shows a different interval for air temperature where each interval represents a separate season. Also the centroid (the center of a cluster) that corresponds to each cluster/season is as such:

- Cluster 0 - Summer: 25.64837027
- Cluster 1 - Winter: 8.09033212
- Cluster 2 - Spring: 17.69155114

Each cluster centroid is comparable to the median of air temperature as shown in **Table [7.2]**.

Cluster 0 represents summer, cluster 1 represents winter, and cluster 2 represents spring. We have performed modeling experiments on each cluster separately and assess their performance.

As been shown in **Table [7.1]**, that the data is clearly separable into 3 different seasons i.e summer, winter, and spring seasons respectively.

Cluster	Season	Data Size
0	Summer	5,504
1	Winter	3,103
2	Spring	5,014

Table 7.1: Cluster by Season

Season	Median TA	Minimum TA	Maximum TA	Common Range
Summer	25.093 C	21.67 C	35.7 C	21.91 C - 26.09 C
Spring	17.91 C	12.89 C	21.66 C	15.36 C - 21.05 C
Winter	8.92 C	-9.68 C	12.88 C	5.74 C - 12.84 C

Table 7.2: Seasons Statistics

As shown in **Table [7.2]**, it is noted that the clusters are clearly separable and this is shown through the distribution of the TA values across the three clusters.

- Cluster 0 has TA values between 21.91 C to 26.09 C which clearly signifies a summer season.
- Cluster 1 has TA values between 5.74 C to12.84 C which signifies a winter season.
- Cluster 0 has TA values between 15.36 C to 21.05 C which signifies a spring season.

Furthermore, we plot the distribution of TA for the whole data. **Figure [7.17]** shows the density plot for TA across all the clusters ranging from -9 to 38

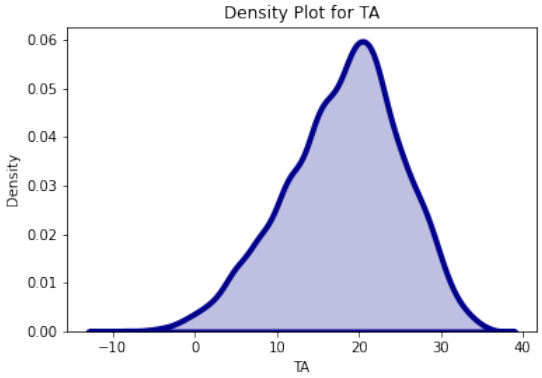
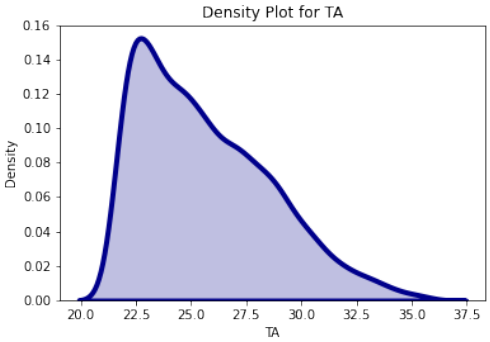
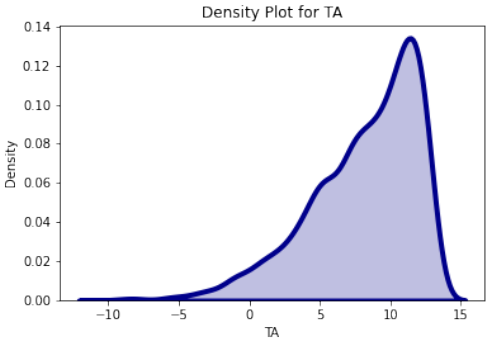


Figure 7.17: Density Plot of TA

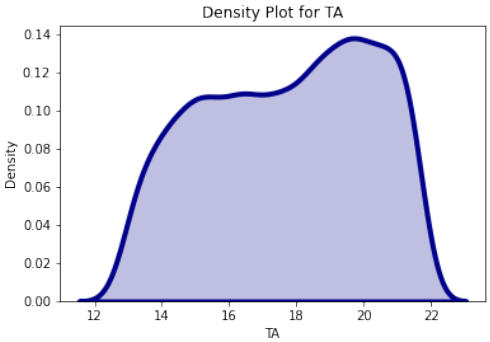
We further zoom in on each cluster and observe the distribution of each season alone.



(a) Summer



(b) Winter



(c) Spring

Figure 7.18: Density Plot of TA Clusters

After inspecting the data and validating that when clustering by TA, the clusters are clearly separable into different seasons, we will further perform modeling on each cluster alone and assess the results in **Chapter [13]**, and **Chapter [15]**

Chapter 8

Utility-Based Regression and Minority Up-Sampling

Our usage of the UBR Metrics is inspired by the authors in the paper (Ribeiro & Torgo, 2008) where they developed the UBR framework which allows evaluating regression models in a cost-sensitive manner leading us to transform our regression problem into a classification-like problem by classifying our output variable as rare versus not rare. The UBR module often estimates the utility of any regression model by attaining a balance between cost and benefits from the obtained predictions by defining a relevance function, rare values, and a relevance function as follows:

8.1 Relevance Function

Relevance Function is domain-dependent and is used to map each target value to a spectrum of 0 to 1 values as 0 being not relevant and 1 being relevant. The most relevant values are also considered rare ie. they do not occur a lot in the data set. This is used to quantify the benefit of a target variable values given that they are not uniform across the variable's domain.

- We separated the data by site Id
- For each site Id, we calculated the percentage of data lying between the following ranges: [1,2], [3,4], [5,6]...[14,15].
- We check the ranges that have only less than 10% data falling into them across all sites.
- We define all real ET values falling between these ranges as rare.
- All values falling from ranges [5,15] are considered rare, with 15 being the rarest.

8.2 Defining Rarity

Having our target variable bounded between 1 - 15 mm, an experiment is conducted to classify the range of rare versus not rare as follows:

- Data is being split by Site Id
- For each Site Id, the percentage of data lying in ranges [1,2], [3,4], [5,6]....[14,15] is computed
- Check the ranges having less than 10% data falling into them
- Classify ranges with less than 10% data as being rare. Others are not rare.

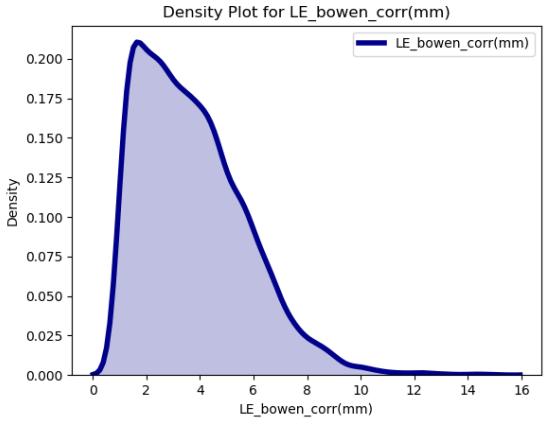


Figure 8.1: Density Plot

All values falling from ranges [5,15] are considered rare, with 15 being the rarest.

Figure [8.1] shows the density plot of LE values ranging between 1 and 15 mm and most data points fall in the lower range.

8.3 Relevance Matrix and Relevance Threshold

Torgo and Rebeiro supported two methods for defining rare values as follows:

- Range: In this type, the user has to define a set of reference points known as the relevance matrix for him to assess relevance. With [1-4] representing being more represented unlike range [5-15]. The user will also have to supply a relevance threshold that ranges from 0-1. We set this value to 0.1

which is used to classify all target values under it as not rare and all the values above this threshold to be as rare.

$$Relevance\ Matrix = \begin{pmatrix} 1 & 0 & 0 \\ 4 & 0 & 0 \\ 15 & 1 & 0 \end{pmatrix} \tag{8.1}$$

- Extremes: In this type, an automated process is used to set the range of rare values depending on the box cox plot graph of the target variable (real ET) allocating higher importance to the least represented target variable values.

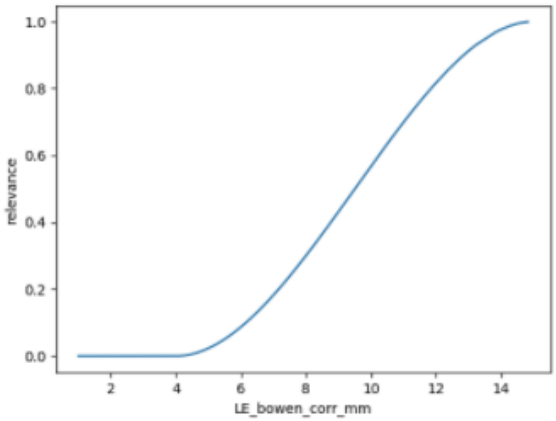


Figure 8.2: Rare versus Not Rare

Figure [**8.2**] represents the relevance values of the points in the data set. As noticed, the relevance values increase as the rarity increases.

8.4 SmoGn

Precision and recall which are acquired from the UBR framework serve something other than estimating the model's accuracy. Precision measures the percentage of relevant results whereas recall measures the percentage of relevant (rare) results that have been correctly classified by our model. We have applied the work of (Branco, Torgo, & Ribeiro, 2017) to obtain higher recall values. Authors in (Branco et al., 2017) propose a SmoGn algorithm that tackles the imbalanced data for any regression problem. We are concerned with the rare values (high values) that are poorly represented in our data set. We will thus use SmoGn to

generate more instances of these values for the model to learn and generalize better and to indicate better decisions for the farmer. SmoGn consists of a random under-sampling technique and two over-sampling ones i.e SmoteR and Gaussian Noise. SmoGn will produce more synthetic information utilizing SmoteR when the chose seed and the K-closest neighbors are close in distance and it utilizes Gaussian Noise when they are not. Moreover, SmoGn also has the option to down-sample the non-rate data samples, but we will rather focus only on over-sampling the rare data. This module offers the user the ability to choose the oversampling percentage, the k for the k-nearest neighbor, and the distance type (Manhattan, Euclidean, etc. . .). We also note that we are applying the SmoGn up-sampling technique on the training data only, not on the validation and testing.

8.5 Utility Based Regression and SmoGn Hyper-parameters

Several hyper-parameters needs to be tuned when incoporating UBR and SmoGn:

- rel_method: relevance method - range or extremes.
- extr_type: extreme type - high or low (meaning if rare values are high or low)
- rell: relevance matrix (specified in the study)
- thr_rel: threshold for relevance
- k: K for K-nearest neighbor
- epl: boolean value controlling the possibility of having a repetition of examples when performing under-sampling by selecting among the "normal" examples.
- dist: distance function used (Manhattan, Euclidean, etc..)
- cperc: A list containing the percentage(s) of under-or/and over-sampling to apply to each "class".
- p: A number indicating the value of p if the "p-norm" distance is chosen.

8.6 Customised K-Fold Cross-Validation

We have incorporated K-Fold cross-validation by using a custom split method i.e we split by site to ensure having the site in each data set.

8.6.1 K-Fold Cross Validation

Ordinarily in an AI cycle, information is partitioned into training and testing sets; the training set is then used to train the model and the test set is utilized to assess the presence of a model. Be that as it may, this methodology may prompt different issues. In more straightforward words, a change issue alludes to the situation where our precision acquired on one test is different to exactness gotten on another test set utilizing a similar calculation.

The solution for this issue is to utilize K-Fold Cross-Validation for execution assessment where K is any number. We partition the information into K folds. Out of the K folds, K-1 sets are utilized for training while the leftover set is utilized for testing. The calculation is trained and tried K occasions, each time another set is utilized as a testing set while remaining sets are utilized for training. Finally, the performance of the model is assessed by averaging the scores across all folds depending on a pre-defined loss.
We will be using 10-fold cross-validation. The data is divided into 10 sets.

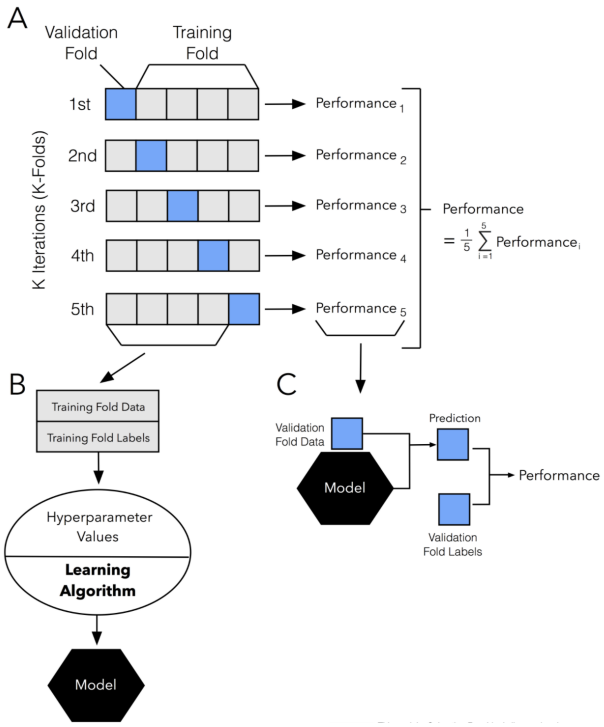


Figure 8.3: KFold Cross Validation

Figure [**8.3**] portrays how the process of k-fold cross validation works for 5-folds.

We use a **GridSearchCV** for it uses a brute force mechanism on finding the best hyperparameters for our model and data set. We have used a grid search

with cross-validation. We give our grid search the following parameters:

- estimator: The model we are using
- param_grid: The parameters of our model and are displayed in the form of a dictionary.
- cv: The cross-validation criteria we are using.
- scoring: The scoring function we need to keep track of from sklearn. metrics. In our case, we use the mean squared error.
- n_jobs: The number of CPU cores to use to train our model. We set this to -1 to use all the available cores.

8.6.2 Split Method

The data set will be split first into 80% train and 20% test and then the train data set is further split into 80% train and 20% validation with having each site being included in each data set. We also performed another split where the TRAIN_RATIO is set to 0.7 and TEST_RATIO is set to 0.3 and analyzed the impact of each split on our model's performance.

Algorithm 9 Split Data

Initialize X_train, X_test, X_valid, Y_train, Y_test, Y_valid
$output\_column$ = "LE_bowen_corr(mm)"
$TRAIN\_RATIO = 0.6$, $TEST\_RATIO = 0.2$
Get unique sites from df
for *site in unique_sites* **do**
    $df\_site = df[site]$
    $X = df\_site$
    Remove output_column from X
    $Y = df\_site$[output_column]
    $train\_index = count(X)$ * $TRAIN\_RATIO$
    $test\_index = count(X)$ * $TEST\_RATIO$
    $X\_train \leftarrow X[:train\_index]$
    $X\_test \leftarrow X[train\_index:test\_index]$
    $X\_valid \leftarrow Y[test\_index:]$
    $Y\_train \leftarrow Y[:train\_index]$
    $Y\_test \leftarrow Y[train\_index:test\_index]$
    $Y\_valid \leftarrow Y[test\_index:]$
end for
return X_train, X_test, X_valid, Y_train, Y_test, Y_valid

In **Algorithm** [**9**], the data set will be split into training, validation, and testing data sets based on the specified TRAIN_RATIO and TEST_RATIO. Each data set will contain records for all the sites we have, in this way our model will train on all the given sites and will better generalize.

Chapter 9

Feature Selection

Feature Selection is a mechanism of reducing the number of features when developing a predictive model to improve the model's performance and to decrease the model's computational cost. The main objective of using feature selection is to identify the input features that are the most relevant to the target variable. Different feature selection approaches will be applied under the scope of a supervised learning problem.
When applying feature selection our aim is as such:

- Reduce over-fitting: Less redundant data means less opportunity for redundant data/noise-based decision making
- Reduce training time: Less knowledge ensures that the algorithm learn faster
- Improve accuracy: Less uncertainty in the data means that the accuracy of the model improves

9.1 Methods

A supervised feature selection method is divided into three groups:

- Intrinsic: It is an approach that attains an automatic feature selection during the training process.
- Filter: It uses statistical techniques that evaluate how a subset of the input features are related to the target variable. Some of them are as follows:
  - Rank all input features by their score and select the top k input features having the highest scores.
    The scikit-learn provides this through **SelectKBest** class.
  - Convert all scores into a percentage metric and select the ones that are higher than a minimum percentile.

- Wrapper: It is a process that selects a subset of the input features based on trying different subsets of the input features and reporting the ones that perform best on a predictive model.

We have applied two different methods for feature selection. A filter-based method based on correlation and mutual information and a wrapper method based on feature importance.

The most common filter-based feature selection techniques for a regression predictive modeling problem with numerical input and output features are Correlation and Mutual Information Feature Selection:

9.1.1 Correlation Feature Selection

Correlation Feature Selection is a feature selection approach that measures how two features change together. An example of such a correlation is Pearson's correlation coefficient that assumes a Gaussian Distribution and often reports if a linear relationship exists.

The scikit-learn provides an implementation of this statistics through **f_regression()** and it is used as an option in **SelectKBest** feature selection strategy in order to select all the features showing a strong linear relationship.

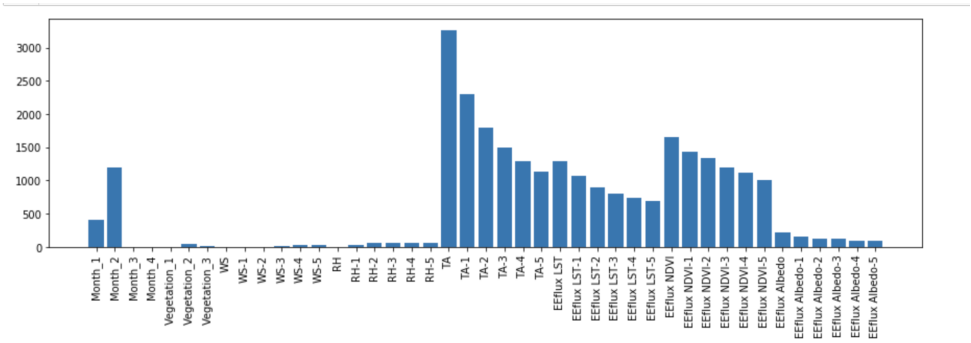


Figure 9.1: Correlation Feature Selection Bar Plot

Figure [9.1] shows the input features on the x-axis and the scores on the y-axis. According to **Figure [9.1]**, it is noticed that TA is the top contributing feature followed by the lags of it, EEflux NDVI, EEflux LST, Month Encoded columns (Month_1, Month_2) and EEflux Albedo.

9.1.2 Mutual Information Feature Selection

Mutual Information is computed between two features measuring the reduction in uncertainty for one given a known value for the other.

It is computed between each input feature and the target variable where it measures the dependency between two variables. MI ranges from 0 to 1, a low MI value indicates no correlation/dependency and a high MI value indicates a high dependency.
It relies on the non-parametric methods being based on entropy estimation and often utilized in non-linear problems.

The scikit-learn library provides an implementation of mutual information for feature selection through **mutual_info_ regression()** and it is used as an option in **SelectKBest** feature selection strategy.

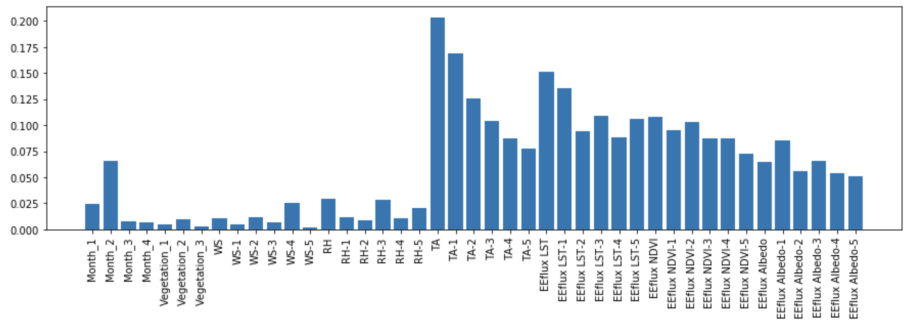


Figure 9.2: Mutual Information Feature Selection Bar Plot

Figure [9.2] shows the input features on the x-axis and the scores on the y-axis. A score always has a positive value bounded between 0 and 1. The higher the score, the better contribution an input feature has on its target.
According to **Figure [9.2]**, it is noticed that TA is the top contributing feature followed by the lags of it, EEflux LST, EEflux NDVI, Month Encoded columns (Month_1, Month_2), and EEflux Albedo.
This method shows almost the same results as that of the correlation feature selection methods in terms of the most contributing input features.

9.1.3 Feature Importance

Feature Importance is a mechanism for assigning a score to the input features depending on how useful each is at predicting the target feature. The computed scores indicate the relative importance of a feature in the prediction process and they are often useful to better understand our model, data, and even in input feature reduction. Moreover, feature importance aid in interpreting our data and can also be used to select and filter out irrelevant features. By using our best model we have obtained the below figure.

As noticed in **Figure [9.3]**, the top contributing features are TA, LST and NDVI.

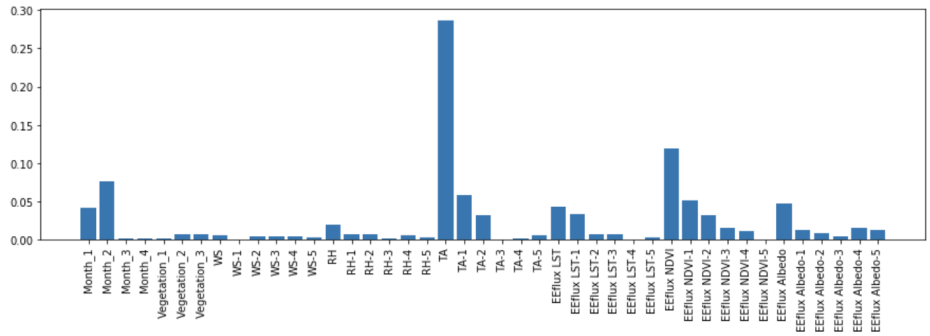


Figure 9.3: Feature Importance

9.2 Scenarios

The feature selection process starts by ranking all input features by Mutual Information Score using a filter-based feature selection algorithm (SelectKBest), then our model will be fed the input features one by one in order of the highest Mutual Information Score.

We also employ Feature Selection methods to reduce the dimensionality of our problem and strike tradeoffs between accuracy, speed of training, and least demanding in terms of input features resulting in the below scenarios:

9.2.1 Scenario A

Scenario A revolves around having a model with all the input features as in without applying any feature selection method. The columns for this scenario are Site Encoded, Month Encoded, Vegetation Encoded, TA + 5 Lags, NDVI + 5 lags, LST + 5 lags, Albedo + 5 lags, WS+ 5 lags, and RH + 5 lags.

9.2.2 Scenario B

Scenario B revolves around having a model that is economically inexpensive with the least training time and least demanding in terms of input features. The columns for this scenario are Site Encoded, Month Encoded, Vegetation Encoded, TA, and 5 of its lags.

9.2.3 Scenario C

Scenario C revolves around having a model that is fed a certain number of features less than the total number of features. The columns for this scenario are Month

Encoded, Site Encoded, TA + 5 Lags, NDVI + 2 lags, LST + 5 lags, Albedo + 2 lags, WS+ 2 lags, and RH + 3 lags.

9.2.4 Scenario D

Scenario D revolves around having a model that is fed columns that have shown to be the top contributing columns to our target variable according to SHAP and as mentioned in **Chapter [16]**. The columns for this scenario are Ta + 5 Lags, LST + 5 lags, and RH + 3 lags.

Name	Input Combinations
Scenario A	All Columns
Scenario B	TA(5 lags)
Scenario C	TA(5 lags) WS(2 lags)
	RH(3 lags)
	EEflux LST(5 lags)
	EEflux NDVI(2 lags)
	EEflux Albedo(2 lags)
Scenario D	TA(5 lags) RH(3 lags)
	EEflux LST(5 lags)

Table 9.1: Feature Selection Scenarios

Chapter 10

Assessment and Evaluation Metrics

We have used regression, correlation and agreement, utility-based regression, and probabilistic model selection metrics for us to evaluate our models.

All error metrics are computed as per scikit-learn

10.1 Regression Metrics

We are quantifying how well our model is performing using different error metrics and accuracy measures.

- Mean Squared Error(MSE): A metric that is mostly used for regression problems. It measures the average of the squared difference between the target variable and its predicted value. Having the squared behavior plays a role in over-estimating how bad the model behaves and is often a preferable metric given the fact that it is differentiable which helps in optimizing it more.

$$MSE = \frac{1}{n}\sum (y - \hat{y})^2 \tag{10.1}$$

  where:
  n = number of observations
  y = array of the target variable values

- Root-Mean-Squared-Error(RMSE): A metric that is mostly used for regression problems. It is the square root of the MSE metric. It is preferable in cases where large errors are undesired since it first squares the error before applying the average.

$$RMSE = \sqrt{\frac{1}{n}\sum (y - \hat{y})^2} \tag{10.2}$$

- Mean-Absolute-Error(MAE): A linear score metric that measures the absolute difference between the target variable and its predicted value. It is often more robust to outliers and doesn't penalize errors as MSE.

$$MAE = \frac{1}{n}\sum |y - \hat{y}| \tag{10.3}$$

- Mean-Absolute-Percentage-Error(MAPE): A metric that measures the percentage of how bad the model performs. It is the percentage of MAE. The least desirable value is 100% and the most desirable is 0%.

$$MAPE = \frac{100}{n}\sum \frac{|y - \hat{y}|}{|y|} \tag{10.4}$$

- Accuracy: A metric that measures the percentage of how well a model performs. It is quite the opposite of MAPE. A high value close to 100 indicates a well-performing model. The least desirable value is 0% and the most desirable is 100%.

$$Accuracy = 1 - MAPE \tag{10.5}$$

- Coefficient of Determination(R^2): A metric that helps in comparing our model's performance to a base-line model, which is the mean of the data, to indicate how well our model behaves in a scale-free matter scaling up to 1. A value close to 1 indicates that the model is performing well.

$$R^2 = 1 - \frac{MSE(model)}{MSE(baseline)} \tag{10.6}$$

- Adjusted R^2: A metric that is similar to R^2, it is often always less than R^2 since it only detects improvements when there is a real increasing predictor rather than improving on increasing terms while the model is not really improving.

$$R_a^2 = 1 - [\frac{n-1}{n-k-1}) \times (1 - R^2)] \tag{10.7}$$

where:
n = number of observations
k = number of independent variables
$R^2{}_a$ = Adjusted R^2

10.1.1 Negative Accuracy and R^2 Score

There exists cases where the Accuracy and R^2 metrics are negative.

R^2 Score

The equation of $R2$ score as defined by scikit-learn is :

$$R^2 score = 1 - \frac{residual\ sum\ of\ squares}{total\ sum\ of\ squares\ (proportional\ to\ the\ variance\ of\ the\ data)} \tag{10.8}$$

where:

$$Residual\ Sum\ of\ Squares\ (SSres) = \sum_i (y_i - f_i)^2 \tag{10.9}$$

$$Total\ Sum\ of\ Squares\ (SStot) = \sum_i (y_i - \bar{y})^2 \tag{10.10}$$

y_i is the actual target variable
f_i is the predicted target variable by the model
$\bar{y}$ is the mean target variable

R^2 score is defined as the proportion of variance explained by the fit. R-square is negative if the fit is worse than just fitting a horizontal line. R^2 score is computed on an assumption that the average line of the target variable is the worst fit a model can have. $SStot$ (total sum of squares) is the squared difference between this average line and original data points. Similarly, $SSres$ (residual sum of squares) is the squared difference between the predicted data points (by the model plane) and original data points. $SSres/SStot$ yields a ratio of how $SSres$ is worse with respect to $SStot$. If our model can build a plane that is comparatively good than the worst, then in most cases $SSres \leq SStot$. It eventually makes R^2 score as positive if you substitute it in the equation. But what if $SSres \geq Stot$? This means that our regression plane is worse than the mean line ($SStot$). In this case, the R^2 score will be negative.

MAPE

The equation of MAPE is as such:

$$\frac{1}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}}{y_i} \right| \times 100 \tag{10.11}$$

A MAPE > 100% means that the errors are much greater than the actual values. For instance, if the actual target variable is a 1, and we predict a 3, the MAPE is 200%. Hence, the accuracy, which is 100-MAPE will be negative.

10.2 Correlation Metrics and Agreement Metrics

We are quantifying how well our model is performing using different correlation measures.

- Pearson Correlation Coefficient: It is used to quantify the strength of the linear correlation between two entities x and y, where the value correlation = 1 means a positive correlation and the value correlation = -1 means a negative correlation. The least desirable value is a 0 and the most desirable is a 1.

$$Pearson = \frac{\sum (x - |\overline{x}|) \times (y - |\overline{y}|)}{\sqrt{\sum (x - |\overline{x}|)^2} \sqrt{\sum (y - |\overline{y}|)^2}} \tag{10.12}$$

- Spearman Correlation Coefficient: Similar to Pearson but does not measure the linear correlation, it measures the monotonic correlation between two entities. "Monotonicity is "less restrictive" than that of a linear relationship." The least desirable value is a 0 and the most desirable is a 1.

$$Spearman = 1 - \frac{6 \sum d^2}{n^3 - n} \tag{10.13}$$

where:
n = data size
d = difference between ranks

- Spatial Correlation Distance: It is the correlation distance between two entities u and v. The least desirable value is a 0 and the most desirable is a 1.

$$Distance = 1 - \frac{(u - \overline{u}) \times (v - \overline{v})}{||u - \overline{u}||_2 ||v - \overline{v}||_2} \tag{10.14}$$

- Normalized Mutual Information (NMI): A metric used to study the agreement between two independent labels when the real ground truth is unknown. It ranges between 0 (no mutual information) and 1 (perfect correlation).

$$NMI(Y, C) = \frac{2 \times I(Y, C)}{[H(Y) + H(C)]} \tag{10.15}$$

where:
Y = class labels
C = cluster labels
H() = entropy
I(Y, C) = mutual information between Y and C

10.3 Probabilistic Model Selection Metrics

- Akaike Information Criterion (AIC): A model selection metric that quantifies the quality of a model with respect to other models. A low AIC is an indication of a better model. It is often prone to over-fitting when adding more parameters which increase the model fitness this is why a penalty term is added for the number of parameters in the model.

$$AIC = n \times \log MSE + 2 \times num\_params \tag{10.16}$$

  where:
  n = number of observations in the training data set
  num_params = number of trainable models which is model dependent

- Bayesian Information Criterion (BIC): It is a metric for model selection, similar to AIC, and can be applied on a finite number of models. Its basis is on the likelihood function but adds a higher penalty.

$$BIC = n \times \log MSE + num\_params \times \log(n) \tag{10.17}$$

10.4 Utility-Based Regression Metrics

When using the utility-based module, we quantify our model's performance according to utility-based metrics as follows:

- Precision: In regression, Precision is a calculated metric that appraises the rare/not rare predictions. The least desirable value is a 0 and the most desirable is a 1 (Ribeiro & Torgo, 2008).

$$Precision = \frac{\sum_{\phi(\hat{y}_i) \geq t_E} \alpha(\hat{y}_i, y_i)\phi(\hat{y}_i)}{\sum_{\phi(\hat{y}_i) \geq t_E} \phi(\hat{y}_i)} \tag{10.18}$$

  where:

  - $\hat{y}_i$ is $y_{predicted}$
  - y_i is y_{actual}
  - ϕ is the relevance function
  - t_E is the relevance threshold
  - α is a function which defines the accuracy of the prediction, where $\alpha(\hat{y}_i, y_i) = I(L_{0/1}(\hat{y}_i, y_i)$
  - $I()$ is the indicator function given 1 if its argument is true and 0 otherwise

– $L_{0/1}$ is a standard 0/1 loss function

- Recall: In regression, Recall computes the portion of events happening in the domain that are caught by the regression model. Recall is a calculated metric that appraises the number of true rare predictions yielded from all rare predictions that should have been made. The least desirable value is a 0 and the most desirable is a 1 (Ribeiro & Torgo, 2008).

$$Recall = \frac{\sum_{\phi(y_i) \geq t_E} \alpha(\hat{y}_i, y_i)\phi(\hat{y}_i)}{\sum_{\phi(y_i) \geq t_E} \phi(\hat{y}_i)} \tag{10.19}$$

- F1: It captures the balance between the precision and the recall. The least desirable value is a 0 and the most desirable is a 1.

$$F1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{10.20}$$

- F2: The F2 score is used as well, in which twice the weight is given to recall as opposed to the weight given to precision. The least desirable value is a 0 and the most desirable is a 1.

$$F2 = 5 \times \frac{precision \times recall}{4 \times precision + recall} \tag{10.21}$$

- F0.5: F 0.5 score is used also in which we give twice as much weight to precision than recall. The least desirable value is a 0 and the most desirable is a 1.

$$F0.5 = 1.25 \times \frac{precision \times recall}{0.25 \times precision + recall} \tag{10.22}$$

Chapter 11

Better Deep Learning Approaches

Configuring a neural network model is considered as a "dark art" because there is no ideal rule that would work for all different problem domains, we cannot just set a model's configuration for a given data set without having to study the type of the problem we have. If we're lucky we can copy the configuration of another network with a similar problem and use them, but this strategy doesn't always lead to good results and we'll more likely work on models that are different than the ones in the literature. This has motivated us to follow tricks and techniques mentioned by Jason Brownlee's Better Deep Learning Book which emphasizes areas for improving the model such as adaptive learning rates, regularization methods, ensemble techniques, etc...
Several techniques are employed to improve our model's weight, performance, and its ability to generalize to new un-seen data by following most of the listed approaches as mentioned in (Brownlee, 2019).
To improve our model in terms of weight, performance, and its ability to generalize to new un-seen data we followed most of the below listed approaches including:

11.1 Better Learning Techniques

These techniques are applied to our neural network model to improve our model's weights in response to our training data set. It revolves around several techniques starting by configuring carefully the capacity of the model, hyperparameters that are related to optimizing the network using stochastic gradient descent algorithm, and to update the weights using the backpropagation of the error. Some of these techniques are as such:

11.1.1 Configuring Capacity

Configuring the model's capacity is performed by varying the number of layers and the number of nodes in the model. We have varied the number of layers of the model by trying out 1 to 5 layers and compared the mean squared error for each case. The average of the target variable is 3.826096.

- Layer 1: MSE 1.509
- Layer 2: MSE 3.952
- Layer 3: MSE 1.683
- Layer 4: MSE 3.955
- Layer 5: MSE 3.952

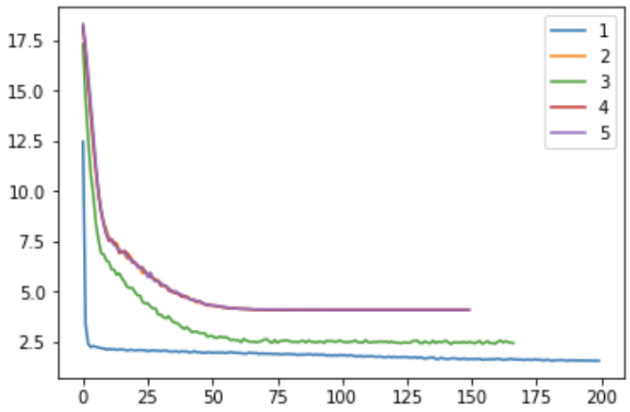


Figure 11.1: MSE for different number of layers

The x-axis shows the number of epochs as the model is being trained on 200 epochs and the reported test MSE by varying the number of layers. We can see that as the number of epochs increases the MSE decreases until the model stabilizes i.e. almost after 55th epochs and layers 1 and 3 are considered better candidates than the others as they have lower MSE and tend to stabilize faster than others.

11.1.2 Configuring Batch Size

We have trained our model using the stochastic gradient descent algorithm which revolves around using the model's current state to make predictions and compute the difference between the predictions and the actual values of our target and then use this as an estimate of the error gradient which is further used to update the model's weight. The number of training examples we use to estimate this

error is the batch size. Since the more training is applied the more accurate the estimate will be, our model performance will be enhanced iteratively, We will vary different batch sizes starting with a small size until we reach the end of our training examples:

1. Batch Gradient Descent: The batch size is set to the total number of training examples. It uses a relatively larger learning rate and more training epochs.

2. Minibatch Gradient Descent: The batch size is set to a size that is more than one and less than the total number of training examples.

In our experiment, we tried a batch size of 32, 64, and to the total number of training samples, i.e. 6,536 and we observe MAE and MSE error metrics.

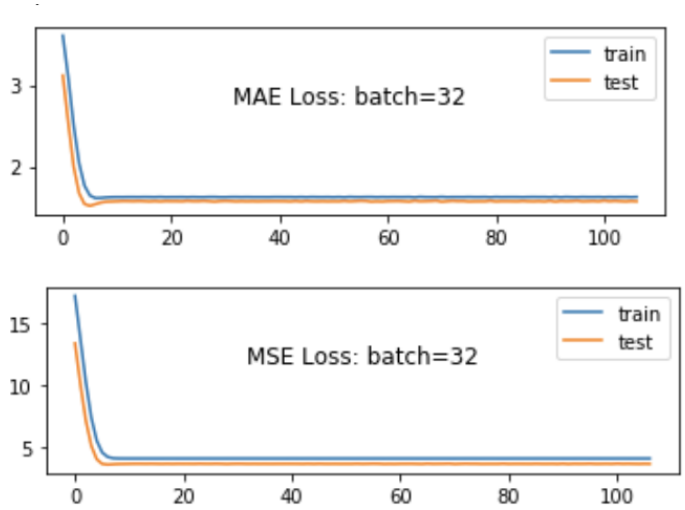


Figure 11.2: MSE for Batch = 32

As it is observed, according to the reported MAE MSE and by visualizing the graphs that the train and validation data error metrics track each other so having a mini-batch gradient descent i.e.(batch size of 32 or 64) is a good candidate. Moreover, it is not worth setting the batch size to be equal to the training data size.

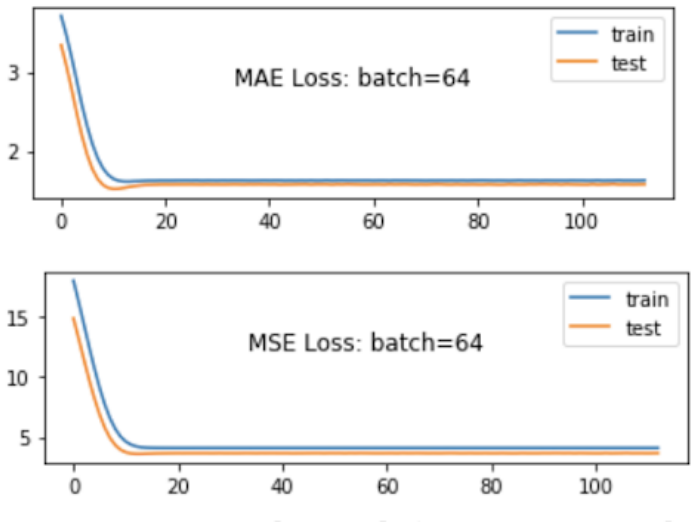


Figure 11.3: MSE for Batch = 64

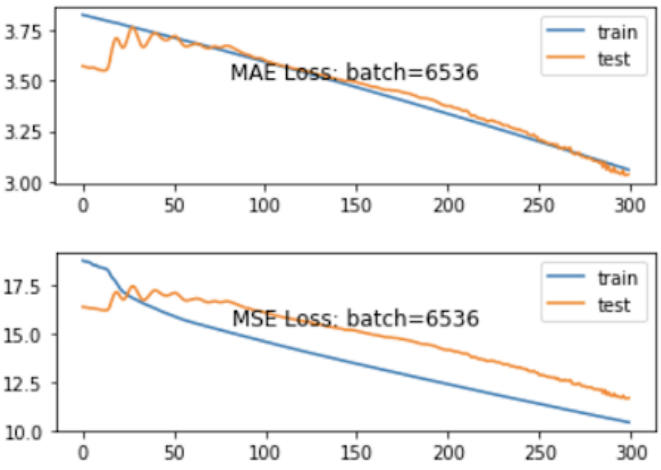
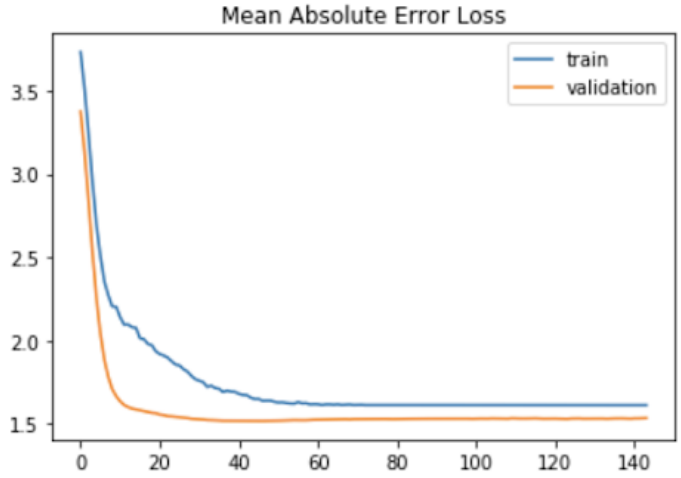
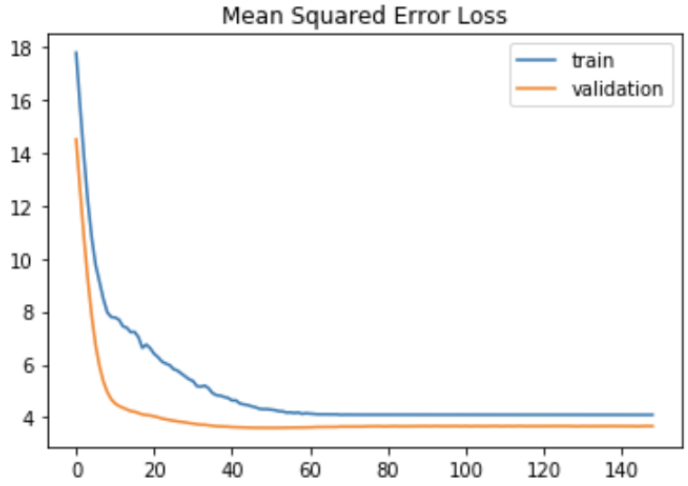


Figure 11.4: MSE for Batch = Data size

11.1.3 Configuring Loss Function

A neural network is trained using a stochastic gradient descent that requires a loss function to calculate the error. We have varied several error functions and compared each.





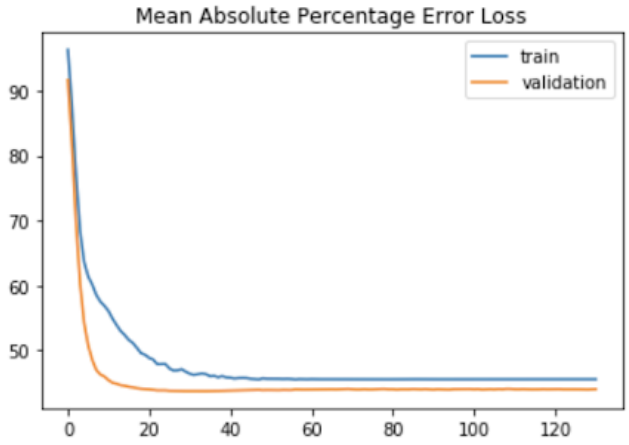
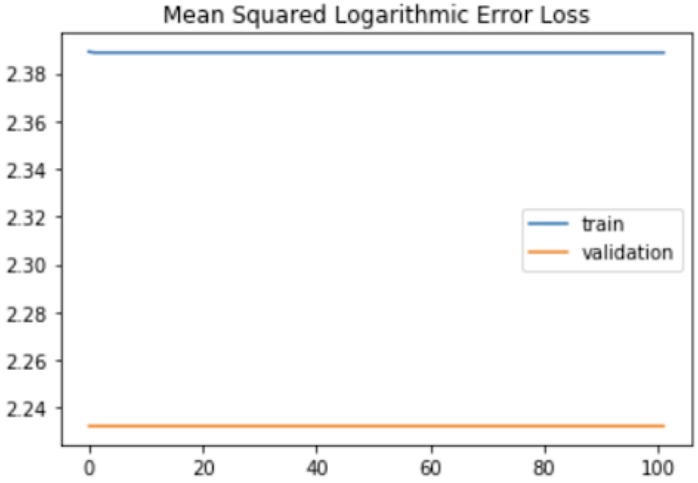


Figure 11.5



It is observed from the learning curves of different loss functions meaning that the squared logarithm error does not generalize well to unseen data, unlike the other candidates.

11.1.4 Configuring Learning Rate

The learning rate is a hyperparameter that shows that the weights in our model are updated the thing that controls the rate at which the model learns. It is a positive value that ranges usually between 0 and 1. The adaptive learning rate would tune the learning rate in response to how the model is learning. Some of the well known adaptive learning rates that are used are Adaptive Gradient Algorithm (AdaGrad), Root Mean Squared Propagation (RMSProp), and Adaptive Momentum Estimation (Adam).

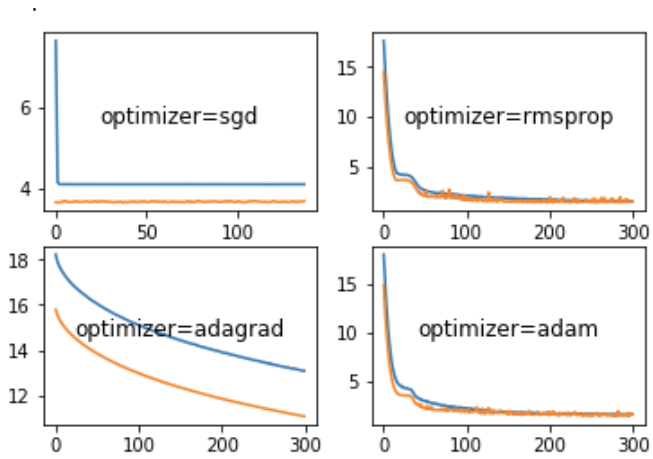


Figure 11.6: Optimizers

We have trained an MLP model for 300 epochs. It is observed from the learning curves that Adam and RMSProp are considered good candidates and others do not generalize well since there is a large gap between the training (represented in blue) and validation (represented in orange).

11.1.5 Applying Data Scaling Approaches

This chapter studies different scaling approaches being applied to the input features and how they affect the network's weight's update procedure and thus affecting the model's performance.

11.1.6 Applying Batch Normalization

We have added batch normalization to our neural network model which often adds stability to the learning process by standardizing the layer's input.

11.2 Better Generalization Techniques

These techniques are applied to our neural network model to improve our model's generalization to unseen/holdout data set and to reduce overfitting a model on the training data set whereby performing very well on the training data set and badly on the testing data set. Those techniques that often aim to reduce the generalization error are referred to as regularization techniques. Techniques vary starting from penalizing a model for having high weights, adding noise, adding drop-out layers, rescaling the weights, etc... The following list summarizes some of these techniques:

11.2.1 Applying Weight Regularization

This approach penalizes the model when having large weights by updating our loss function to encourage smaller weights and thus lower's the model's complexity. There are several approaches to penalize the model, one of which is to calculate the size of the weights by using the L1 or L2 norm. L2 or weight decay is often used more in the neural network field. We experimented with different values of varying the kernel regularizer of an L2 norm. A line plot showing the MAPE between the train and test data sets is created. The MAPE starts high and then lowers at value = 10-4 to its lowest. However, as we increase the weight regularization parameter values, the MAPE increases back. However, as we increase the weight regularization parameter values, the MAPE increases back.

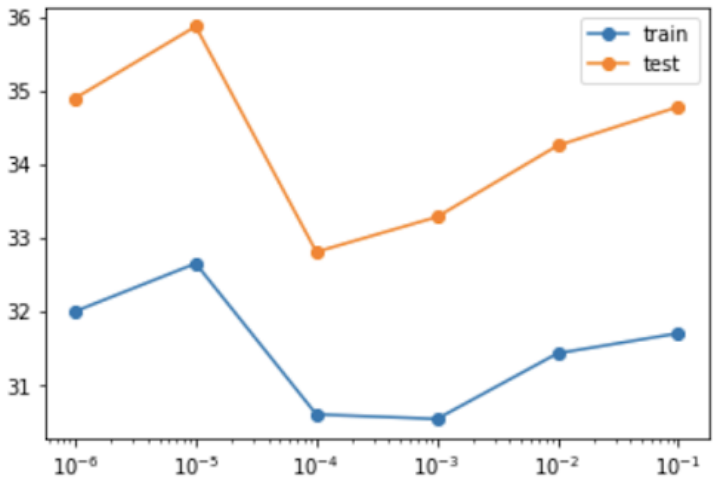


Figure 11.7: Weight Regularization Learning Curve

11.2.2 Activity Regularization

This approach penalizes the activations of the units of the neural network model to encourage more sparse representations. We experimented with different values of an activity regularizer using an L1 norm and tracked MAPE between the train and testing data sets, we notice that 10^{-2} is not a good candidate for our model.

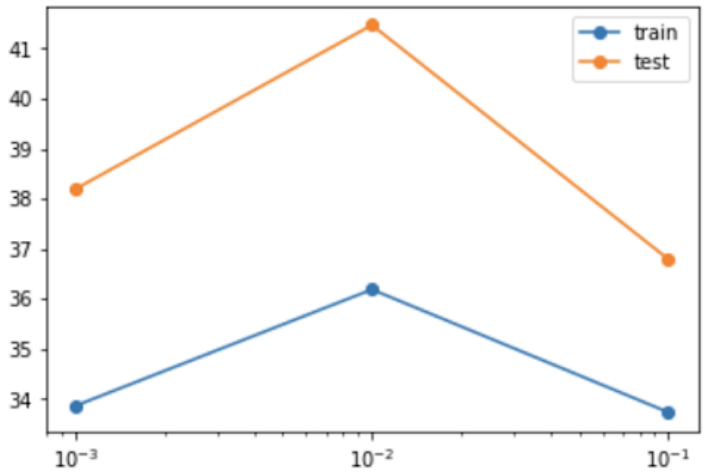


Figure 11.8: Activity Regularization Learning Curve

11.2.3 Forcing Small Weights With Weight Constraints

This approach updates the model's weight by rescaling each when its norm exceeds a pre-defined threshold. We have compared two experiments: In the first experiment, we didn't add any constraint to our model. Then, in the second experiment, we added a unit norm as a kernel constraint to penalize our model. We computed MSE and MAPE for the training and validation data sets. W noticed that we did not get that significant improvement from adding the constraint. However there is a small improvement i.e. at the end of the experiment, the test MSE improved from 1.65 to 1.42 and MAPE improved from 37

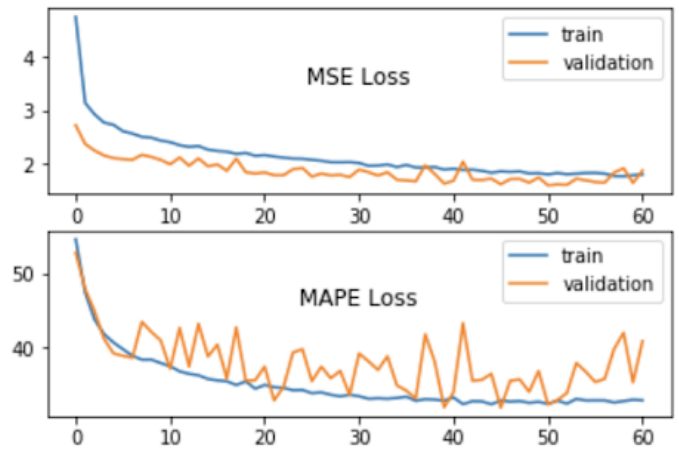


Figure 11.9: No Constraint

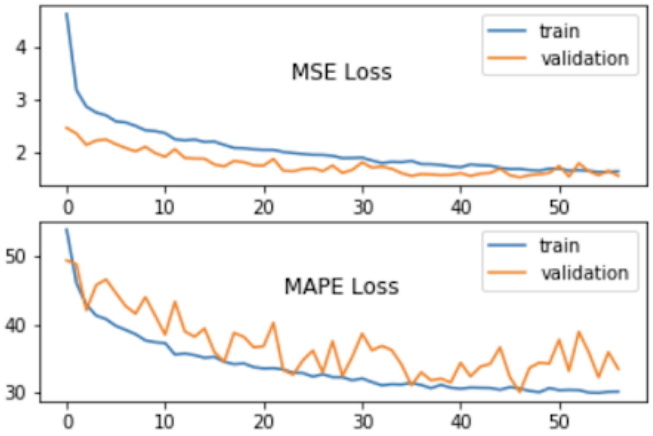


Figure 11.10: Unit Norm Constraint

11.2.4 Adding Drop out Layers

Adding a drop out often leads to excluding randomly connections/weights while training the network causing to decrease in the tight coupling between network nodes. This should be a value between 0 and 0.5. We experimented with different drop-out values ranging from 0.1 to 0.5 and we compared MAPE between the training and testing data sets. We note that we do not have an over-fitting problem in the data set we are tackling but as it is observed that the MAPE value is at its lowest at dropout=0.2.

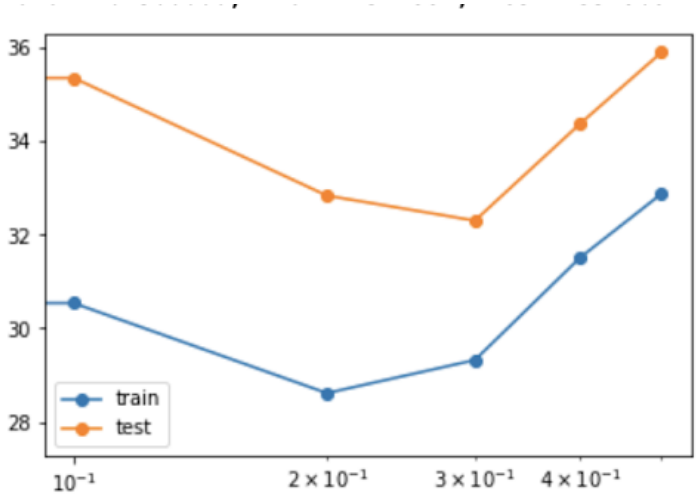


Figure 11.11: Dropout Variations Learning Curve

11.2.5 Adding Noise

This approach will often cause to decrease in some dependence on the input by introducing a statistical variation to the input layer. We did not have to add any noise to our data set as we did not see any need for it.

11.2.6 Early Stopping

This approach is used to monitor the model's performance on the holdout data set while training and stops when the performance starts to degrade. By this, we can monitor any error metric. Early stopping callback is being introduced which will track in our case the validation MAPE per epoch with a patience value of 10. Once triggered the model will stop training but will wait with the patience of 10 epochs before stopping if no improvement was observed. This is done since we will not benefit to proceed further because we do not see any improvement and would rather see worse results or no improvements afterward. However, the model at the end of the training phase might not be the best i.e. the one having the lowest MAPE on the validation data set, this is why we add another callback which is the ModelCheckpoint which will save the best model observed during the training phase depending on the measure we choose to observe on the validation data set.

After having to apply better deep learning approaches and tunning hyperparameter using a grid search, we come up with the best hyperparameters that we have used in our final experiments.

- The number of hidden neurons: This number represents the model's capacity to learn, having a more complex model requires a higher number of neurons, but a too high number will cause the model to memorize the data set and overfit. The number of hidden neurons is set to 64.

- Activation function: A function that is attached to each neuron in the network and decides if a neuron should be fired or not based on its input. if this input happens to be relevant to the model's prediction. It also aids in mapping the output to a range between 0 and 1 or -1 and 1. The activation function is set to softmax.

- kernel initialization: This is used for setting the initial weights to be uniform.

- Kernel regularizer: We have added an L2 norm kernel regularization with a value of 0.001

- Batch Normalization: A mechanism for adjusting and scaling the input to speed up the learning process. It reduces the amount by which the hidden unit values shift around.

- Drop out: A regularization technique that revolves around randomly dropping neurons so that their weights will not be updated resulting in improving generalizing and decreasing a model's ability to overfit. It ranges from 0 to 0.5. We set this to 0.4.
- The number of layers: The number of hidden layers in our model between the input and output layers. This is set to 2.
- Optimizer function: A function that is defined to reduce the loss by updating the weights or learning rates. We set it to Adam which is an adaptive learning algorithm that often adapts or learns the learning rate itself.
- Loss function: The loss function compares the model's predicted value against the grounded truth. We set it to mean squared error which is often used for a regression problem.
- Epochs: The number of iterations the training data set is being shown to the neural network model while training. This is set to 500 with monitoring validation mape to stop early if the validation scores start to drop.
- Batch Size: A mini-batch size is the number of samples that we give to our model after the parameter update occurs. The batch size is set to 64.
- Scaling: Data is being normalized using MinMax Standardization with a range of 0, 1.

11.3 Better Predictions Techniques

These techniques are applied to our starting neural network model to decrease the variance in the performance of our final model. The variance can be reduced by combining predictions from different models which are referred to as ensemble learning. In other words, we introduce bias to reduce our model's variance. Ensemble learning also often produces better predictive performance or at least ensures consistent predictive results. There are several techniques to form an ensemble, i.e. we can vary the training data per member, vary the members that contribute to the ensemble, vary the way we combine predictions from these members, and so on. Some of the techniques are as follow:

11.3.1 Model Averaging Ensemble

A neural network model often is prone to overfitting and suffers from high variance. This issue could be further addressed by training several models and combining their predictions. This approach is known as model averaging where we end up averaging the predictions we obtained. In this chapter, we experimented

with 5 MLP models with the same best hyperparameters configurations we have used and recorded the different error and correlation metrics for each. Then, we computed the average scores across them.

Error Metrics	ensemble_0	ensemble_1	ensemble_2	ensemble_3	ensemble_4	ensemble_avg
Average	3.747154763	3.747154763	3.747154763	3.747154763	3.747154763	3.747154763
R2	0.63	0.58	0.63	0.64	0.64	0.64
Adjusted R2	0.63	0.57	0.62	0.64	0.63	0.64
MAE	0.87	0.97	0.88	0.87	0.87	0.87
MSE	1.44	1.66	1.45	1.4	1.43	1.4
RMSE	1.2	1.29	1.2	1.18	1.19	1.18
Spearman	0.79	0.78	0.79	0.8	0.79	0.8
MAPE	29.46	34.83	30.78	30.59	29.85	31.102
Accuracy	70.54	65.17	69.22	69.41	70.15	68.898

Figure 11.12: Error Metrics for different ensembles

Hence, the final average score for R2 is 0.64 and accuracy is 68.89% which is better than running an individual model.

11.3.2 Weighted Average Ensemble

When applying a modeling averaging ensemble technique the predictions are being combined from each model equally without giving any emphasis to the more skillful models, however, In the weighted average ensemble approach we add a weight to each model's prediction to indicate its trust. In this chapter, we trained 5 MLP models with the same hyper-parameters. Each time we grab a subset of these models and compute the test metrics of this ensemble subset. We vary the ensemble subset starting from one member until we take all members. Then, we compare these metrics to that of the one formulated after training on each model separately. It can be seen that the single model's MAPE score is lower than the ensemble but the ensemble MAPE score is more precise. The orange line plot represents the ensemble scores and the blue line plot represents each model's score.

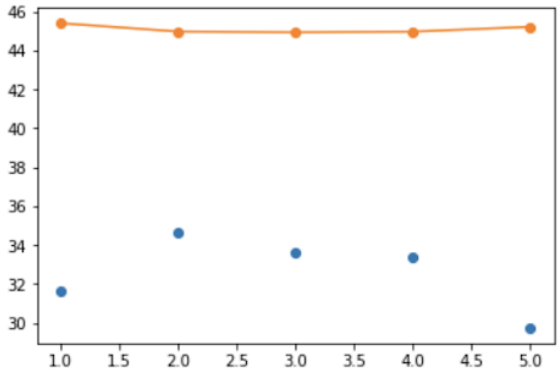


Figure 11.13: Weighted Average Single versus Ensemble Model

11.3.3 Re-sampling Ensemble

An effective model requires members to disagree and vary with their prediction errors causing them to be good in different ways. To achieve this difference between models, we train each on a different subset by using one of many re-sampling techniques i.e. cross-validation, random splits, or bagging.

- Random Splits: Data is repeatedly and randomly divided into training and testing data sets. We trained an MLP model with 10 splits and in each split, then we randomly divided the train data set into train and validation respectively by 80% 20%. Finally, we recorded some error metrics.

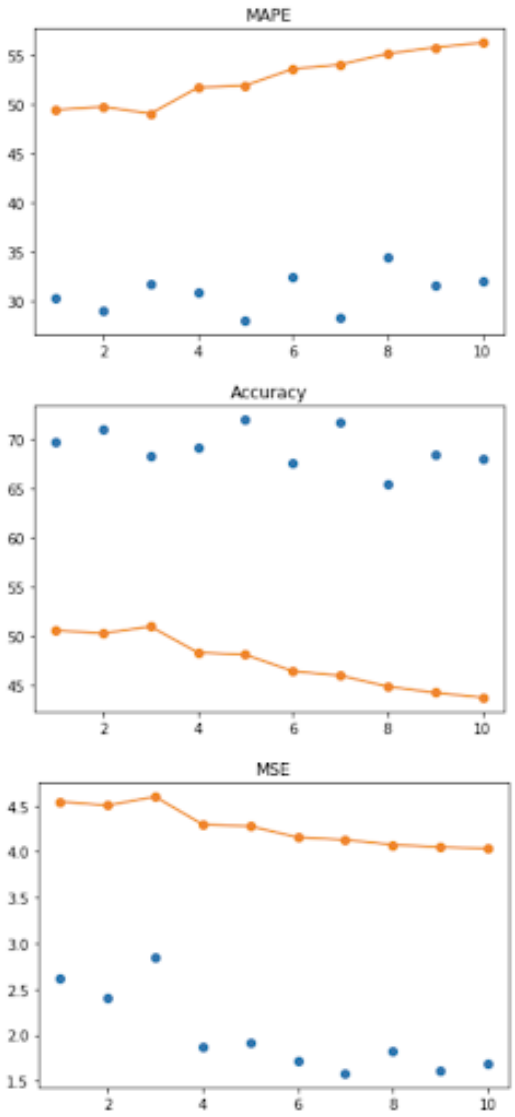


Figure 11.14: Random Split

- Cross Validation: Data is divided into k-folds. Each fold acts as a holdout set where others are being used for training We train an MLP model with k-fold=3 and in each fold we split the train data into train and validation data sets and recorded some error metrics. We notice that as we train more subsets we get better scores for an ensemble. An ensemble of 3 models achieves the lowest MAPE, but the score of a single model is much better compared to that of an ensemble, and the reported average MAPE = 29.280 with a standard deviation of 0.579.

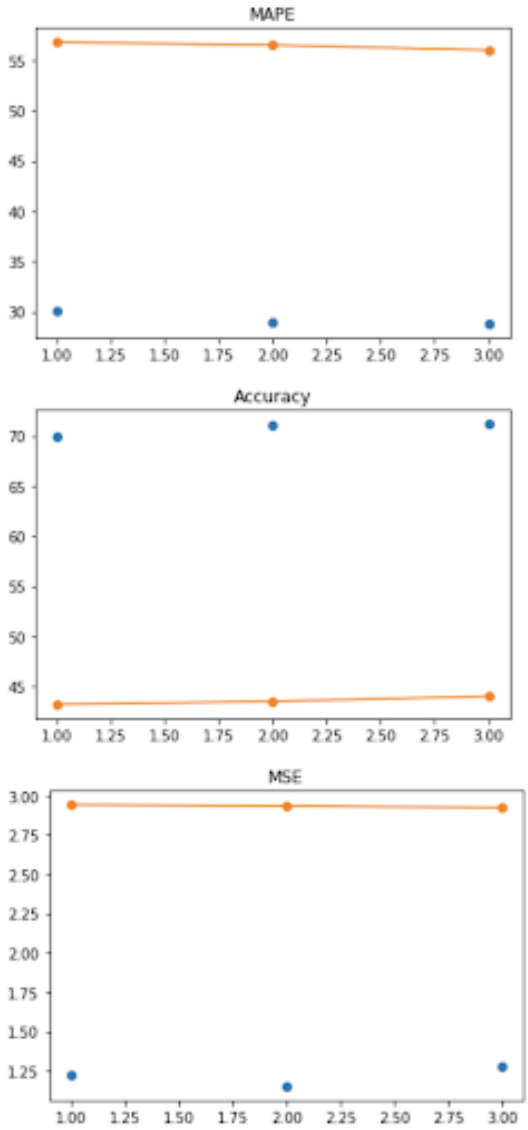


Figure 11.15: Cross Validation Split

- Bagging: Data is being randomly sampled with replacement and the rest is used for testing. We trained an MLP model with 3 splits and in each split,

we re-sample data with replacement (bagging) and divided the data into a train data set and a validation data set. Also, we recorded some error metrics. An ensemble of subset 1 achieves the lowest MAPE, but the scores

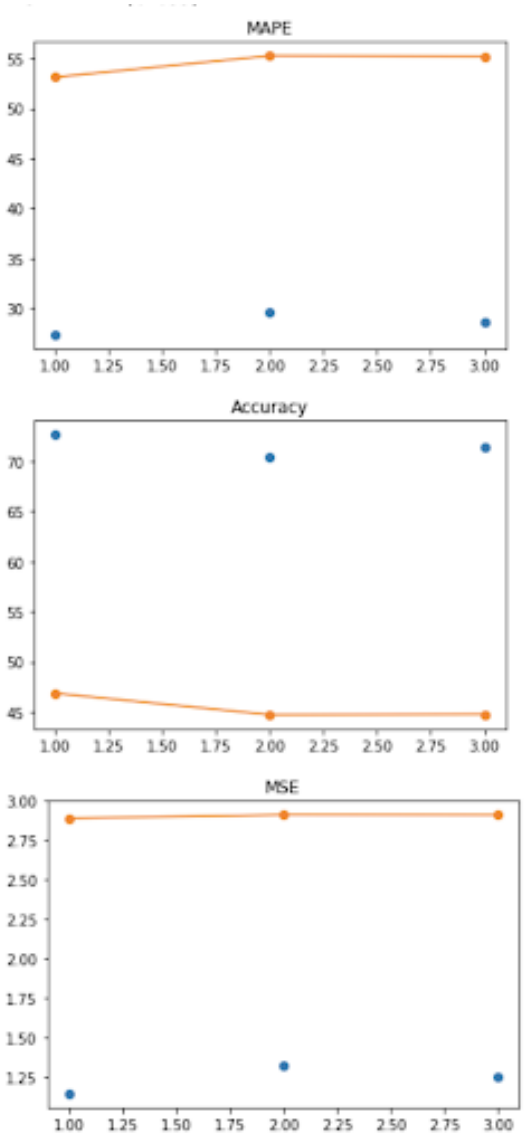


Figure 11.16: Bagging

of a single model are much better compared to that of an ensemble, and the reported average MAPE = 28.544 which seems better than when using cross validation.

11.3.4 Horizontal Ensemble

When having a small number of training data compared to unlabeled data, the model often suffers from high variance, and it's hard to figure out which is the best model to use. This has motivated the introduction of an approach named horizontal ensemble. It is a method that revolves around using several models being members of an ensemble and saving their results at the end of a contiguous block of epochs before the end of the training phase and then averaging their results. This would result in better performance than randomly choosing one as our final model. This method is proposed by Jingjing Xie, et al. in their 2013 paper Horizontal and Vertical Ensemble with Deep Representation for Classification. The following is done:

- We select models that are trained for a relatively stable range of epochs.
- The resulted predictions are then averaged.

We have trained an MLP model on 400 epochs and then started to save the model results while the epoch varies from 350 until the end. In total, 50 models were saved. These 50 models are being used to form an ensemble. We then compared the results we got out of the ensemble versus a single model. The orange represents the ensemble and the blue dots represent the single scores. it is noted that the scores for the ensemble are much better than that of a single model. Also, they stabilize more as we increase the number of epochs and predict more test data.

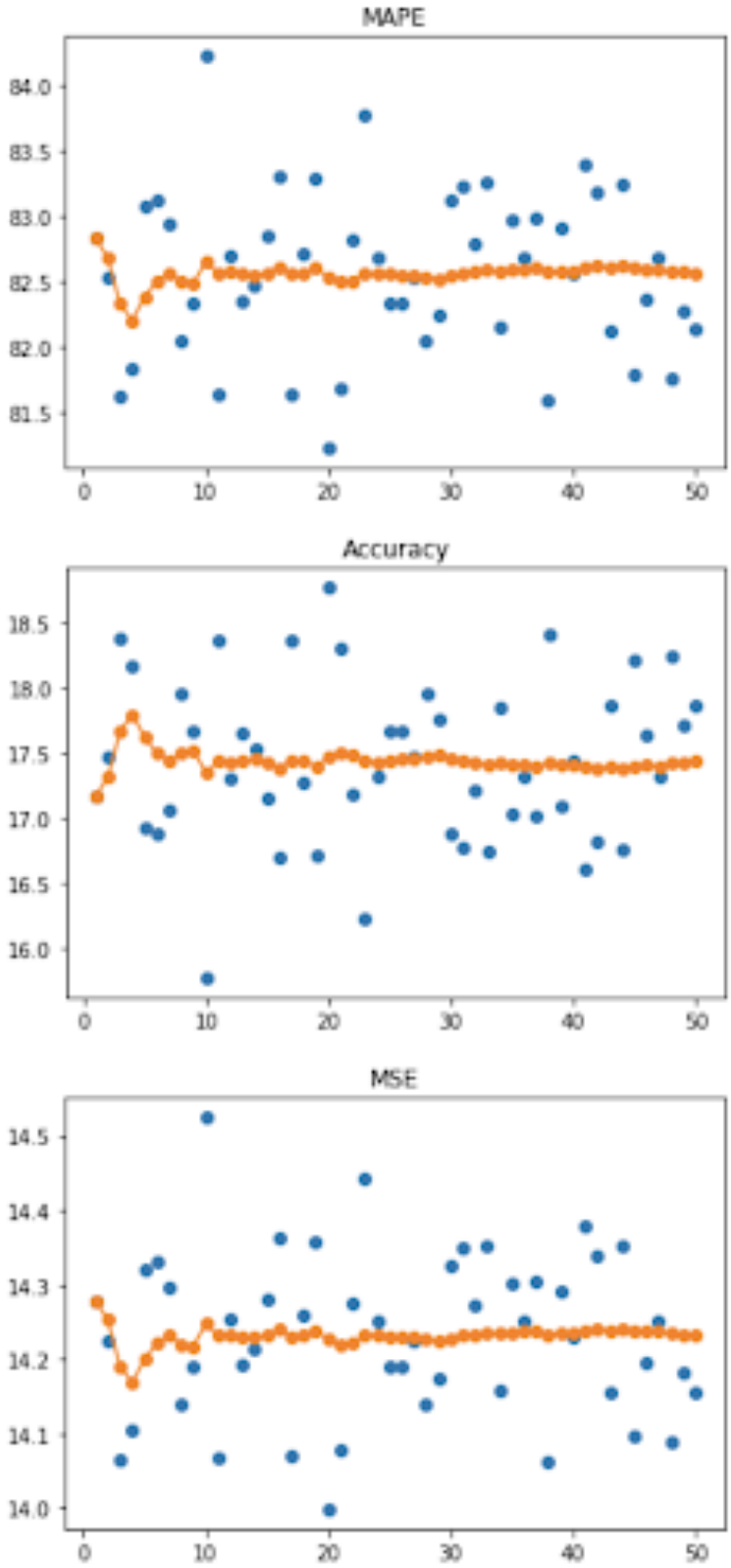


Figure 11.17: Horizontal Ensemble

11.3.5 Cyclic Learning Rate Snapshot Ensemble

Forming an ensemble is often better than relying on a single model since it has a lower generalization error but we often suffer from computational cost. For that matter, we could train multiple snapshot models during a single run and form an ensemble, but this will lead to similar prediction errors. Also, we need an approach to have diverse skills for each which is why we use aggressive learning. An aggressive learning rate schedule involves changing the learning rate over the training epoch called cosine annealing. This methodology requires specifying the initial learning rate and the total number of training epochs. It has the impact of starting with a large learning rate that would rapidly decrease towards the end before increasing again. This causes the model's weights to change dramatically. Initially, the weights are more likely to be good which results in having diverse skills per model or snapshot. Each time the model converges, we save the snapshot and add it to the ensemble.

11.3.6 Stacked Generalization Ensemble

Stacked Generalization or Stacking is an approach where we create a completely new model by combining predictions out of sub-models. This is often used as a replacement of only averaging the results of the model's which often have the same contribution. It constitutes of 2 levels:

- Level 0: In this level, we make predictions out of the training data.
- Level 1: In this level, we take the output from level 0 to be our input and we learn how to make predictions from this data. In this level, we denote the model to be a meta-learner.

11.3.7 Average Model Weight Ensemble

In this approach, we combine the weights from several models into a single model for making predictions which would result in a more stable and better-performing model. Models are being created with either equal, linearly, or exponentially weighted average of the parameters out of multiple models as follows:

- We load 5 previously trained MLP models.
- For each model, we get the weights of its layers.
- Compute the average of all the layer weights
- Loop over a subset of the models.
- Assign an equal, linear, or exponential weight for each subset.

- Evaluate error metrics on the test data for each subset.
- Evaluate error metrics on the test data for every single model.
- Compare the above two steps.

We experimented with building an ensemble of 5 Multi-layer perceptron models. Each model has the same hyper-parameters. However, we are varying the learning rate that we are passing to the model starting with a large number i.e 0.1. We deployed an aggressive learning rate schedule that involves changing the learning rate over the training epoch which further forces a change in a model's weight. Then, we average the result of the scores we got. We often got more consistent and stable results when combining several ensemble strategies rather than obtaining random predictions out of a single model.

Chapter 12

Experimental Variations

12.1 Experiment A

Real ET versus ET (predicted or obtained by EEflux METRIC). We denote this experiment by Experiment A. Error metrics are calculated between Variable 1 and Variable 2:

- Variable 1: Real ET
- Variable 2: Predicted ET or EEflux METRIC ET

High R2 (close to 1) and low error metrics indicate that our model can predict Real ET well.

Research Question: Which model best estimates Real ET?

12.2 Experiment B

The proportional residual between Real ET and Predicted ET is computed by getting the error metrics corresponding between two entities:

- Variable 1: EEflux ET / Real ET
- Variable 2: EEflux ET / Predicted ET

We denote this experiment by Experiment B.

Research Question: Which model best minimizes the proportional bias between the Real ET and EEflux ET?

12.3 Experiment C

The absolute residual between Real ET and Predicted ET is computed by getting error metrics and other accuracy measures corresponding between two variables:

- Variable 1: EEflux ET - Real ET
- Variable 2: EEflux ET - Predicted ET

We denote this experiment by Experiment C.

Research Question: Which model best minimizes the absolute bias between the Real ET and EEflux ET?

12.4 Experiment D

A combination between the proportional and the absolute residual. The combination residual between Real ET and Predicted ET is computed by getting the error metrics corresponding between two entities:

- Variable 1 : (EEflux ET - Real ET) / Real ET
- Variable 2 : (EEflux ET - Predicted ET) / Predicted ET

We denote this experiment by Experiment D.

Research Question: Which model best minimizes the combined bias between the Real ET and EEflux ET?

Chapter 13

Point-wise Modeling

Point-wise Modeling is a single estimated prediction made by the trained model for a given entry.

A set of point-wise regression models were evaluated starting with a base model i.e a Vanilla Multi-layer Perceptron Model (MLP) to an Ensemble of MLP reaching a Meta-learning Model.

We have applied several better deep learning, generalization, and ensemble techniques for improving the model's weight, performance, and its ability to generalize to new unseen data such as adaptive learning rates, regularization methods, ensemble techniques. etc.. to our models as mentioned in **Chapter [11]**.

Data is further split into 70/30 ratio and utility-based regression techniques are employed.

13.1 Models

We experimented with the following Point-Wise Prediction Models:

13.1.1 Vanilla Point-wise BDL Model

Multi-layer perceptron model (MLP) is denoted by Vanilla Point-wise BDL Model. It is a neural network model and a supervised learning algorithm that learns a function given a data set.

13.1.2 Ensemble of MLP

We experimented with building an averaged ensemble of five MLP models by varying the learning rate and deploying cosine annealing for scheduling the learning rate with the same best hyper-parameters for each MLP model.

13.1.3 Meta-Learning

Meta-learning is a process of learning how to learn. We learn an initialization for network parameters in such a way that the network is able to adapt to new tasks quickly.

13.1.4 Auto ML

Auto ML Table is a supervised learning service that trains a machine learning model with the data provided using the standard machine learning workflow which consists of data preparation, model training, evaluation, deployment, and prediction. It defines your problem and model based on the type of the target column, if it contains numerical data, then a regression model will be used otherwise a classification model. This tool is integrated with Google Cloud AI Platform where it internally tries several deep learning models and chooses the best based on MSE. We have used AutoML for the Manual Bowen daily data set and trained on 70% of the data size. The best model that AutoML selected was a DNNLinear model. Furthermore, it also outputs the feature importance for all the input features scoring the highest for the air temperature feature followed by Month encoded, EEflux NDVI, RH, etc...

13.2 Choosing the Best Model

We have performed Experiment A on all our point-wise models which were trained on all the input features.

	Metrics	Experiment A
Models	**Average ET**	3.74715476
	Train Size	8,729
	Test Size	2,182
	Recall	0.80873
	F2	0.81749
	R2	0.63
	RMSE	1.21
MLP	**MAE**	0.9
	Accuracy	69.5
	NMI	1
	Training Time (seconds)	637.84
	Testing Time (seconds)	0.124
	Recall	0.941
	F2	0.944
	R2	0.619
	RMSE	1.22
Ensemble MLP	**MAE**	0.88
	Accuracy	68.7
	NMI	1
	Training Time (seconds)	755.32
	Testing Time (seconds)	0.78
	Recall	0.96005225
	F2	0.96157961
	R2	0.79383871
	RMSE	0.9088138
MLP-Reptile	**MAE**	0.67232353
	Accuracy	76.6164768
	NMI	0.99512033
	Training Time (seconds)	1665.2541
	Testing Time (seconds)	97.1628394

Figure 13.1: Point-wise models

As shown in **Figure** [**13.1**], we note that MLP is the least performing model with an R2=0.63 and Recall=0.80 hence we call it our base model and MLP-Reptile is the most performing model with an R2=0.79 and Recall=0.96 hence we call it our best model.

We will measure the percentage of improvement between variable a and variable b and it will be used in the below sections as such:

$$percentage\ of\ improvement = \frac{metric_a - metric_b}{metric_a} \tag{13.1}$$

where metric_a represents the value of the metric we are measuring for model a and metric_b represents the value of the same metric we are measuring but for model b. An example of a metric we are studying is R2.

13.3 Base Model

Our base model is a Vanilla Point-wise BDL Model. It is composed of at least 3 layers of nodes: An input layer, a hidden layer, and an output layer. It relies on a technique known as backpropagation which performs a backward pass that aims in minimizing a loss function between the actual target and its predicted value by tuning the input's weights.

13.3.1 Architecture

We train our regression problem using a neural network that consists of two hidden layers with a softmax activation function each. Both hidden layers consist of 64 hidden neurons followed by an output layer. A dropout layer is added between each hidden layer to reduce over-fitting and the model is further trained on 500 epochs with the loss function being mean squared error. This is our base model that is tuned based on applying different better deep learning techniques as mentioned above.

13.3.2 Hyper-parameters

We tuned a set of hyper-parameters for they gave the highest accuracy and lowest error metrics. They are as follows:

- The number of hidden neurons: This number represents the model's capacity to learn, having a more complex model requires a higher number of neurons, but a too high number will cause the model to memorize the data set and overfit. The number of hidden neurons is set to 64.

- Activation function: A function that is attached to each neuron in the network and decides if a neuron should be fired or not based on its input if this input happens to be relevant to the model's prediction. It also aids in mapping the output to a range between 0 and 1 or -1 and 1. The activation function is set to softmax.
- kernel initialization: This is used for setting the initial weights to be uniform.
- Kernel regularizer: We have added an L2 norm kernel regularization with a value of 0.001.
- Batch Normalization: A mechanism for adjusting and scaling the input to speed up the learning process. It reduces the amount by which the hidden unit values shift around.
- Drop out: A regularization technique that revolves around randomly dropping neurons so that their weights will not be updated resulting in improving generalizing and decreasing a model's ability to overfit. It ranges from 0 to 0.5. We set this to 0.4.
- The number of layers: The number of hidden layers in our model between the input and output layers. This is set to 2.
- Optimizer function: A function that is defined to reduce the loss by updating the weights or learning rates. We set it to Adam which is an adaptive learning algorithm that often adapts or learns the learning rate itself.
- Loss function: The loss function compares the model's predicted value against the grounded truth. We set it to mean squared error which is often used for a regression problem.
- Epochs: The number of iterations the training data set is being shown to the neural network model while training. This is set to 500 with monitoring validation MAPE to stop early if the validation scores start to drop.
- Batch Size: A mini-batch size is the number of samples we give to our model after the parameter update occurs. The batch size is set to 64.
- Scaling: Data is being normalized using MinMax Standardization with a range of 0, 1.

13.3.3 Implementation

We have built a multi-layer perceptron neural network and we have applied a variety of better deep learning, generalization, and ensemble techniques by employing a grid search for tuning hyper-parameters and incorporating utility-based

learning. The data set is divided into 70% training and 30% testing without replacement and the training data set is further divided into training and validation data sets.

This module outputs validation scores, standard deviation validation scores, test scores, in addition to all necessary plots, figures, and data sets needed and found here

This module code is implemented using Keras. Our implementation is available in here and consists of the following python script:

- keras_neural_network_ubr.py: The main script file that contains the neural network architecture, hyper-parameter tuning, and utility-based regression integration.

13.4 Best Model

Meta-Learning has recently emerged as a methodology for learning from small amounts of data by learning from a variety of related tasks. It seeks to optimize a fast-learning algorithm using a data set of tasks. Each task is a regression or classification problem with a certain distribution (same problem in different domains). Each task is composed of training and testing data sets. (Yin, 2020) Our algorithm will be fed a training data set and therefore, will produce an agent that will produce a good average performance on the testing data set. We aim to validate as has been proposed by authors in (Finn et al., 2017) if we can train a model that can easily adapt to a new task having only a few data points. We will further validate that using a real-world data set of a 2D structure rather than what is implemented by the authors in (Finn et al., 2017) as a sine wave regression problem of 1D.

Moreover, will meta-learning trained with the same number of examples and the same architecture of our base model (MLP) beat our MLP point-wise model?

In summary, meta-learning does not only learn to solve a new task, however, it learns to solve many tasks by learning each time a new task. i.e upon experience (learns to learn).

13.4.1 Meta-Learning Approaches

A target few-shot task is handled either using a metric-based approach or an optimization-based approach.

Learning to embed: Metric-Based Meta-Learning

Metric-based Meta-Learning (metric learning) learns a distance function between data points to categorize the test instances based on the K labeled examples. The distance function is composed of two steps: one is an embedding function that encodes instances into a certain space and the other is a similarity metric i.e Euclidean distance or cosine similarity that is used to compute how close any two instances are in the space. If we were able to learn the distance well on the training tasks, then the model will generalize well on the target task without further fine-tuning. This approach is not concerned with updating a network's weights. Examples of networks/models using the metric-based approach are Siamese Network, Machine Network, etc... This has been mentioned in detail (Wenpeng Yin, 2020).

Learning to fine-tune: Optimization-Based Meta-Learning

Optimization-Based Meta-Learning works by learning a good initial parameter initialization for a neural network model using only a few samples and few gradient steps to reach an optimal point for a new unseen task. We have performed several experimental evaluations using a family of meta-learning algorithms that use the optimization-based approach (Nichol, Achiam, Schulman,2018) as such:

- Model Agnostic Meta-Learning (MAML): A meta-learning algorithm that learns by learning from a variety of related tasks consisting of only a small number of data points and then the meta-learner will produce a quick learner that will be able to generalize well on new related tasks.
- Reptile: A first-order meta-learning algorithm that is somehow similar to MAML and works by sampling a task repeatedly and moving the initialization towards the trained weights on a task.
- FOMAML v2: An approximation to MAML by ignoring second-order derivatives. The computational cost of it is only related to computing gradient descent since FOMAML is a first-order MAML algorithm.

13.4.2 MAML Algorithm Explained

A model-agnostic meta-learning was proposed by (Finn, Abbeel, Levine, 2017). It learns initialization parameters that often allow the model to adapt efficiently and quickly to an unseen task with only a few samples. It is agnostic in the sense that it can be applied to any neural network model. MAML is considered hard to train since it involves two levels of training: the meta-backpropagation implies the computation of gradients of gradients (second-order derivatives) (Wenpeng Yin, 2020).
It consists of the following steps per episode/iteration:

Given: A model f with parameter θ having a distribution over tasks $p(T)$ where

- θ: Denotes the initial weight vector parameter for the model that we aim to tune
- θ': Denotes the optimal weight vector parameter for the model for a specific task
- T: Denotes a task and each task could represent a regression or classification problem with a certain distribution. It is composed of training and testing data sets that are often assumed to come from the same distribution of tasks $p(T)$.
- $p(T)$: Denotes the probability distribution of a task
- Episodes: Represents the number of iterations or the number of different combinations of tasks we use to train our model. We perform a grid search in order to choose the number of episodes/iterations per experiment.

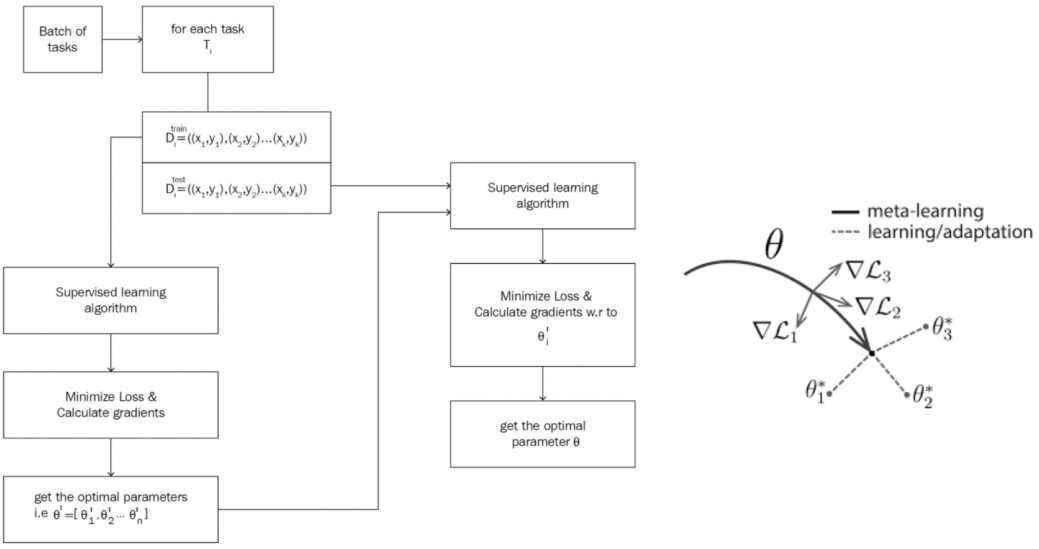


Figure 13.2: MAML Algorithm from (Finn et al., 2017)

1. We randomly initialize θ.

2. Sample a batch of tasks T_i from a set of tasks
   i.e. $T_i \sim p(T)$, assume we have n tasks T= $\{T_1, T_2, T_3.....T_n\}$

3. Inner Loop:

   - For each T_i, we sample k data points to prepare the training and testing data sets
     - $D_i^{train} = (x_1, y_1),(x_2, y_2).....(x_k, y_k)$: is used in the inner loop to find the optimal parameters θ'_i
     - $D_i^{test} = (x_1, y_1),(x_2, y_2).....(x_k, y_k)$: is used in the outer loop to find the optimal θ
   - We apply a supervised learning algorithm on D_i^{train} and calculate the mean squared error loss using gradient descent to get the optimal parameters
     $$\theta'_i : \theta'_i = \theta - \alpha \nabla_\theta L_{T_i}(f_\theta) \tag{13.2}$$
     for each task resulting in n optimal parameters i.e. $\{\theta'_1, \theta'_2, \theta'_3....\theta'_n\}$ as mentioned by (Finn et al., 2017) where:
     - α: Represents the step size hyper-parameter
     - ∇_θ: Represents the Gradient descent
     - $L_{T_i}(f_\theta)$: Represents the loss function for task T_i, In our regression problem, the loss function is mean squared error.

4. Outer Loop: Meta – Learner

   - We minimize the loss by calculating the gradient descent with respect to our optimal parameter θ'_i in the test data set D_{test} as shown in **Figure [13.2]**
   - We update our randomly initialized parameter θ using the test data set.

5. Repeat steps 2 to 4 for some n number of episodes (iterations/epochs) until our solution converges to an optimal solution where the mean squared loss starts to minimize reaching a very low value without further increasing or the number of episodes is reached.

13.4.3 Reptile Algorithm Explained

A first-order optimization-based meta-learning algorithm (Nichol, Achiam Schulman, 2018) that is similar to MAML given both are model-agnostic and rely on meta-optimization through gradient descent. It works by sampling a task repeatedly and moving the initialization towards the trained weights on a task.

Given:

- θ: Denotes a vector of parameters for the model
- T: Denotes a task
- L: Denotes any loss function, in our case we use mean squared error to be our loss function given our problem is a regression problem
- k: Denotes the number of gradient steps we seek to reach an optimal solution using gradient descent.
- SGD(L,θ, k) Denotes the function i.e (stochastic gradient descent) that performs k gradient steps on loss L starting with θ and returns the final parameter vector
- Episodes: Denotes the number of iterations or the number of different combinations of tasks we use to train our model.

1. Initialize θ
2. For each iteration/episode
   (a) Sample a task T in any approach in such a way all tasks belong to the same probability distribution. Each task is a learning problem (regression or classification problem) with a loss L_T (Nichol, Achiam, & Schulman, 2018)
   (b) Compute the weights W of our neural network using SGD
   (c) Update θ in the direction of $W - \theta$ or using stochastic gradient descent or Adam being an adaptive learning rate optimization algorithm designed for training neural networks.

13.4.4 Difference between MAML and Reptile

Both seek an initialization for the parameters of a neural network having that the network will be fine-tuned using a small amount of data only from a new task. MAML uses the gradient descent algorithm, whereas Reptile performs stochastic gradient (SGD) without having to compute the second derivative. As a result, Reptile is less expensive than MAML (Nichol & Schulman, 2018).

MAML also optimizes the efficiency of an algorithm on the support/testing set ensuring the learned algorithm can learn fast in the few-shot samples of the target task. In contrast, Reptile works by optimizing the system to work well on all the training tasks and may work well for the case the training tasks are close to the target tasks. (Wenpeng Yin, 2020). **Figure [13.3]** summarizes the difference between MAML and Reptile.

MAML	Reptile
It is considered computationally expensive to train since it involves two levels of training: the meta-backpropagation implies the computation of gradients of gradients.	Less memory and computational wise as compared to MAML
It unrolls and differentiates through the computation graph of the gradient descent algorithm	It performs a stochastic gradient without having to unroll a computation graph or compute the second derivative.

Figure 13.3: MAML versus Reptile

In order to decide on which algorithm to use for us to proceed with our experiments, we have run an experiment comparing MAML and Reptile algorithms and reported the best algorithm on our daily data set.

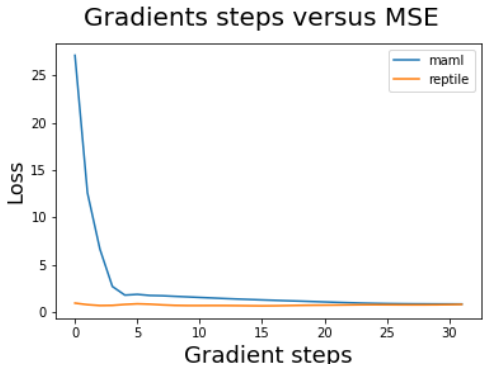


Figure 13.4: Gradient steps versus MSE

Figure [13.4] shows the gradient steps on the x-axis and the loss being mean squared error on the y-axis.

It is observed that as we increase the gradient steps, the loss decreases in both MAML and Reptile. However, Reptile converges much faster than MAML.

Thus our best model is a Multi-layer perceptron (MLP) model that is based on the Meta-Learning Reptile algorithm. We denote it by MLP-Reptile.

13.4.5 Meta-Learning and Conventional Approaches

- In an conventional approach, tasks are often solved from scratch using a learning algorithm unlike meta-learning where we learn from other tasks.
- In an conventional approach, models do not usually perform well when the data is expensive or rare to collect or even when compute resources are unavailable. However, meta-learning can learn using only few-shot samples.

- In an conventional approach, the initial weights of a model are not optimized as opposed to optimization-based meta-learning where the initialization is optimized internally resulting in having a better head-start than traditional models.

13.4.6 Meta-Learning and Multi-tasking

Multi-tasking is a process that learns all of the tasks simultaneously faster than learning them independently. This approach is considered effective when the tasks share some commonality. However, meta-learning is a process that learns a new task more quickly given prior experience or knowledge on previous tasks even for learning in different domains. Thus multi-tasking can be considered a form of meta-learning.

13.4.7 Meta-Learning and Transfer Learning

Transfer Learning uses the knowledge gained from a source task to improve learning on a target task by pre-training a parameter prior and optional fine-tuning (Task A helps Task B). Meta-Learning however, refers to a methodology that aids in improving transfer learning. It assumes that the training and target tasks come from the same distribution meaning they refer to the same problem but represent two different domains. This strict assumption is not adopted by transfer learning since it can pre-train on any source task that is helpful to that of a target task.

Transfer learning becomes handy when we do not have data and we want to generalize to a field that is similar to ours. However, meta-learning is considered a better candidate when sparse data and no existing pre-trained models.

13.4.8 Architecture

We train our regression problem using a neural network that consists of two hidden layers with a RELU activation function each. The hidden layers consist of 40 and 60 hidden neurons respectively followed by an output layer.

13.4.9 Hyper-parameters

The hyper-parameters we tuned for MLP-Reptile are as follows:

- Number of layers: The number of hidden layers that compose the neural network other than the input and output layers.

- Number of neurons for each layer: The number of hidden neurons for each layer.

- Activation: The activation function used in each hidden layer is ReLU.
- Loss: The loss function we are optimizing is the mean squared error.
- Optimizer: The optimizer used is an adaptive optimizer which is Adam.
- Inner Learning rate: The learning rate that we use for the training process which we feed to our optimizer.

We vary the parameters that are related to the meta-learning task:

- Episodes/Tasks: It represents the number of iterations or the number of different combinations of tasks we use to train our model. Each task could correspond to a different input combination.
- Gradient steps: It represents the number of gradient steps we seek to reach an optimal solution.

13.4.10 Implementation

We have built a Reptile Meta-Learner and employed hyper-parameters tuning and utility-based learning. The data set is divided into 70% training and 30% testing.

This module outputs testing scores, in addition to all necessary plots, figures, and data sets needed and found here

This module code is implemented using Tensorflow. Our code is inspired by the Github code which is available here, but we have further tuned the scripts to match our implementation of providing a dynamic data set of two dimensions.

Our implementation is available here and divided into several files as such:

- dataset_preparation.py: This script contains the definition of the data set and some processing helper methods.
- TaskGenerator.py: This script contains the generation of the training and testing data sets.
- MAML.py: This script contains the definition of the MAML model.
- FOMAML.py: This script contains the definition of the FOMAML model.
- Reptile.py: This script contains the definition of the Reptile model.
- maml_reptile.py: This script contains the code for training all our defined models on different tasks, on the training data set, and then evaluating the testing data set.

- maml_reptile_climate.py: This script contains the code for training on different climates and defining them to be our training tasks and then evaluating on a new climate.

We ran the testing experiment 100 times to produce stable and consistent results each time on a different subset. This is not a hyper-parameter to tune, this is just a number set to 100 as per (Kiat, 2018) where the authors simulated a 1D random sine wave problem with 10000 episodes. This is comparable to our data size which is almost 10,911 rows.

We tuned the MLP-Reptile hyper-parameters by utilizing a grid search and Utility-based regression. Each range of values for each hyper-parameter is chosen based on what is accepted in the literature and based on applying different better deep learning techniques as mentioned in **Chapter [11]**.

- Number of layers: [1, 2, 3, 4, 5]
- Number of neurons per layer: [40, 64, 128, 256]
- Inner Learning rate: [0.001, 0.01, 0.1, 0.2, 0.3]
- Episodes: [500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000]
- Gradient steps (k): [1, 32, 64]

The best hyper-parameter combination yielding the highest Accuracy and lowest error metrics is:

- Number of layers: 2
- Number of neurons for the first layer: [40]
- Number of neurons for the second layer: [64]
- Inner Learning rate: [0.01]
- Episodes: This value is dependent on the experiment performed for each climate/cluster. The best hyper-parameter for each experiment is summarized in **Table [13.1]**

Experiment/Climate	Csb	Dsa	Other	Csa	Union of Clusters	Cfa	Cwa
Sampling Randomly	500	500	3000	3000	3000	3000	3000
Sampling By Climate	1500	1500	1500	1500	3000	1500	1500

Table 13.1: Number of Episodes per Climate

- Gradient steps (k): [1]

Experiment/Season	Spring	Winter	Summer	Union of Clusters
Sampling by Season	3000	3000	3000	3000

Table 13.2: Number of Episodes per Season

The best UBR and SmoGn parameters are:

- rel_method = "range"
- extr_type = "high"
- coef = 1.5
- relevance_pts = np.array([[1, 0 , 0],[4, 0 , 0], [15, 1 , 0]])
- thr_rel=0.1
- Cperc = np.array([1,1.2])
- repl = False
- dist = "Manhattan"
- p = 2
- pert = 0.1

More details about each UBR and SmoGn parameters are found in **Chapter** [**8**]

13.4.11 Tasks Definition

We have performed a variety of experimental variations and varied our training and testing tasks and further evaluated the impact of these variations by either splitting our data randomly, by climate, or by season.

A task represents a shuffled single data set. We define our training and testing tasks as follows:

1. Option 1 (Sampling Tasks Randomly):
    - Training Task: A training task represents 70% of our data set. In this sampling option, we use one training task.
    - Testing Task: A testing task represents the other 30% of our data set. In this sampling option, we use one testing task.
2. Option 2 (Sampling Tasks by Climate):

- Training Task: A training task represents a single climate from **Table [13.3]**. In this sampling option, we use five training tasks.
- Testing Task: A testing task represent an unseen climate from **Table [13.3]**. In this sampling option, we use one testing task.

3. Option 3 (Sampling Tasks by Season):

- Training Task: A training task represents a single season from **Table [7.1]**. In this sampling option, we use one training task.
- Testing Task: A testing task represent the same season from **Table [7.1]**. In this sampling option, we use one testing task.

Sampling Tasks Randomly

We shuffled our data set and took 70% to be as our training task and 30% to be as our testing task. We have varied the number of episodes/iterations we used to run our experiments and evaluated our testing data by running them on 100 iterations as been defined in (Finn et al., 2017) and since it is a reasonable number to ensure we achieve consistent testing results. Moreover, tasks are also shuffled per iteration. For this split, we performed several experimental variations on all the daily data, and each climate alone.

Sampling Tasks By Climate

We have shuffled our data set and took five of the climates as being referred in **Table [13.3]** to be our training tasks and the remaining climate to be our newly unseen testing task. We have further performed a variation of these experiments in terms of testing each time on a new climate and increasing the number of episodes for each of the training tasks/climates.

Climate	Description	Data set Size
Cfa	Humid subtropical	3645
Cwa	Humid subtropical	674
Csa	Mediterranean	3509
Csb	Mediterranean	615
Dsa	Dry Continental	807
Other	Unlabeled/Unclassified	1666

Table 13.3: Climates

The reason for this split is to study the impact of testing our meta-learner on a completely new climate and to see if a worse performing climate can benefit out of a good performing climate which often emphasizes the power of transfer learning.

Sampling Tasks By Season

After performing several clustering techniques, we are concerned with identifying if our data is separable by air temperature on a certain cluster/season. We thus perform a statistical and modeling assessment on clustering by air temperature (TA) using Kmeans with k=3. We note that clustering by TA has shown a clear clustering separability portraying seasonal representation of each cluster.

Cluster 0 represents summer, cluster 1 represents winter, and cluster 2 represents spring. We have performed modeling experiments on each cluster separately and assessed their performance. More details about the seasonality study is available in **Chapter [7]**

13.5 Results

We will present the testing results for all the sampling techniques we have performed and further analyze each. We have conducted Experiment A across MLP-Reptile, MLP, EEflux (METRIC), AutoML, and Stat on our daily data (union of clusters) and on different sampling techniques. We have conducted Experiment A on all the mentioned models across different feature selection scenarios (scenario A, scenario B, scenario C, and scenario D). However, we will first compare MLP-Reptile before and after Up-sampling in order to proceed with the best experimental setup.

13.5.1 Utility-Based Learning and SmoGn Up-sampling

We have performed Experiment A on our best point-wise model (MLP-Reptile) that is obtained from **Section [13.2]** and compared the result of this experiment to when we apply SmoGn upsampling to our data set. SmoGn upsampling is applied to our best model (MLP-Reptile). Details for the SmoGn hyper-parameters are found in **Section [13.4.10]**.

Figure [13.5] portrays the experimental results for the union of clusters (all the data set) before SmoGn and after SmoGn using an MLP-Reptile model. MLP-Reptile with no SmoGn is performing better than MLP-Reptile with SmoGn by an R2 of 5.13%, Accuracy of 6.94%, Recall of 0.52%, and F2 of 0.07%. The reason for that is because Meta-Learning is by itself powerful even with few shot examples, so no need for further upsampling in our case. Upsampling is adding only more noise to our data set and thus the results are not as good as before applying SmoGn.

Takeaway Message: Applying SmoGn to our MLP-Reptile model was not as efficient where MLP-Reptile with SmoGn yielded in a decrease in R2 by 5.14%, in

	Metrics	Union of Clusters - Before SmoGn	Union of Clusters - After SmoGn
Models	Average ET	3.74581288	3.74581288
	Train Size	7638	7638
	Test Size	3,273	3,273
	Recall	0.96005225	0.96512443
	F2	0.96157961	0.96088148
	R2	0.79383871	0.75509963
	RMSE	0.9088138	1.00634559
MLP-Reptile	MAE	0.67232353	0.76596618
	Accuracy	76.6164768	71.6423255
	NMI	0.99512033	0.99485215
	Training Time (seconds)	1665.2541	2548.1658
	Testing Time (seconds)	97.1628394	134.313482
	Data size	10,911	10,911

Figure 13.5: Before and After SmoGn

Accuracy by 6.94%, in Recall by 0.52%, and in F2 by 0.07% as opposed to without up-sampling. SmoGn has created noisy data samples which caused our model's performance to degrade. Thus we continue with using MLP-Reptile without any SmoGn techniques for any further experiments.

13.5.2 Sampling Tasks Randomly

We have conducted Experiment A across several models on our daily data (union of clusters) and on different clusters that are shuffled randomly with a split ratio of 70/30. The best feature selection scenario yielding the best Accuracy, Recall, and the least error metrics is "Scenario C". We will show the results for that scenario and all other scenarios will be presented in the appendix.

We have performed seven experiments where each experiment represents training and testing on the same climate. The number of episodes for each cluster is summarized in **Table [13.1]** for the sampling randomly approach. Experiments are defined as such:

- Cluster Csb: We randomly train on 500 episodes using a 70/30 split of climate Csb
- Cluster Dsa: We randomly train on 500 episodes using a 70/30 split of climate Dsa
- Cluster Other: We randomly train on 3000 episodes using a 70/30 split of climate Other
- Cluster Csa: We randomly train on 3000 episodes using a 70/30 split of climate Csa
- Union of Clusters: We randomly train on 3000 episodes on 70% of the data and test on 30% of the data for all the climates

- Cluster Cfa: We randomly train on 3000 episodes using a 70/30 split of climate Cfa
- Cluster Cwa: We randomly train on 3000 episodes using a 70/30 split of climate Cwa

	Clusters	Cluster Csb	Cluster Dsa	Cluster Other	Cluster Csa	Union of Clusters	Cluster Cfa	Cluster Cwa
Models	**Average ET**	2.4071607	2.6196219	3.50686273	4.00808472	3.74581288	4.00774363	4.51677034
	Train Size	431	565	1167	2457	7638	2552	472
	Test Size	184	242	499	1,052	3,273	1,093	202
	Recall	0.0001	0.0001	0.0001	0.0001	0.78	0.73	0.0001
	F2	0.0001	0.0001	0.0001	0.0001	0.78	0.72	0.0001
	R2	-30.577819	-1.4356181	-0.888912	-0.4214216	-0.9572085	-1.8242985	-0.4214216
	RMSE	3.82441468	1.37960202	2.51340802	2.35727511	2.79025128	3.33013862	2.35727511
EEflux (METRIC)	**MAE**	2.38679044	1.2173055	2.02203571	1.54465131	2.1026953	2.39641085	1.54465131
	Accuracy	NA	41.8801079	42.6522932	69.4002766	39.1700276	33.7737912	69.4002766
	NMI	0.98412698	1	0.86805742	0.98870571	0.90721809	0.82271656	0.98870571
	Training Time (seconds)	NA	NA	NA	NA	NA	NA	NA
	Testing Time (seconds)	NA	NA	NA	NA	NA	NA	NA
	Recall	0.0001	0.94021794	0.96386927	0.97080728	0.96005225	0.96732238	0.97984997
	F2	0.0001	0.9529583	0.96234943	0.96431185	0.96157961	0.96722553	0.98129906
	R2	0.52397249	0.63417245	0.72493756	0.73748261	0.79383871	0.85639626	0.88049752
	RMSE	0.64055749	0.8603871	0.97749839	0.99249297	0.9088138	0.77587707	0.50704663
MLP-Reptile	**MAE**	0.51605497	0.6399777	0.75571992	0.75410042	0.67232353	0.58284988	0.39129795
	Accuracy	73.2309573	70.4912265	70.0605258	74.8355267	76.6164768	80.9146655	90.4018855
	NMI	1	1	0.99021936	0.98981888	0.99512033	0.99929661	0.99935315
	Training Time (seconds)	373.96572	401.5446	600.85776	1016.22102	1665.2541	950.77944	486.321
	Testing Time (seconds)	107.2686	112.074272	104.385039	115.58596	97.1628394	167.007247	110.067949
	Recall	0.00001	0.89798946	0.88680246	0.92673028	0.9259478	0.9297036775	0.9497509582
	F2	0.00001	0.00004	0.9110401	0.93378058	0.9378146	0.9406024293	0.00004
	R2	0.51104046	0.40678114	0.51188491	0.60148922	0.65130298	0.6795279101	0.5571167
	RMSE	0.65575748	1.14578606	1.40731786	1.22366767	1.17380524	1.195869171	0.98225045
MLP	**MAE**	0.52224957	0.84170251	1.00921214	0.87698342	0.82042826	0.8242660495	0.75835168
	Accuracy	74.4137471	63.4869533	63.5327184	74.3498917	74.328515	74.58439234	76.6873862
	NMI	1	1	1	1	1	1	1
	Training Time (seconds)	285.46254	168.297828	131.949494	156.614623	260.793119	190.669224	425.939502
	Testing Time (seconds)	1.27572989	0.9590857	0.16196299	0.55923653	1.1848011	0.58198786	0.68710041
	Recall	-	-	-	-	-	-	-
	F2	-	-	-	-	-	-	-
	R2	-	-	-	-	-	-	-
	RMSE	-	-	-	-	-	-	-
Stat	**MAE**	-	-	-	-	-	-	-
	Accuracy	-	-	-	-	-	-	-
	NMI	-	-	-	-	-	-	-
	Training Time (seconds)	-	-	-	-	-	-	-
	Testing Time (seconds)	-	-	-	-	-	-	-
	Recall	-	-	-	-	-	-	-
	F2	-	-	-	-	NA	NA	-
	R2	-	-	-	-	0.736	0.801	-
	RMSE	-	-	-	-	1.012	0.988	-
Auto ML	**MAE**	-	-	-	-	0.71	0.706	-
	Accuracy	-	-	-	-	75.95	77.37	-
	NMI	-	-	-	-	NA	NA	-
	Training Time (seconds)	-	-	-	-	7200	7200	-
	Testing Time (seconds)	-	-	-	-	NA	NA	-
	Data Size	615	807	1666	3,509	10,911	3,645	674

Figure 13.6: Experiments for Random Sampling

Figure [13.6] portrays the experimental results across different climate clusters and the union of clusters. We compare the performance of our four models: EEflux (METRIC), MLP (base model), MLP-Reptile (best model), and Auto ML. We note that MLP-Reptile is performing the best among all others in predicting Real ET across all cluster variations, producing an R2 score of 0.79, an RMSE of 0.90, and an Accuracy of 76.6% on the union of clusters. Transfer learning is also highlighted when training the model on the union of clusters. All the four models performed poorly on some climate clusters (Dsa, Csb), but excellently on the others (Other, Csa, Cfa, Cwa). This implies that the models have learned from well-performing clusters yielding more accurate predictions on the others. The clusters are ordered from worst to best in terms of their results.

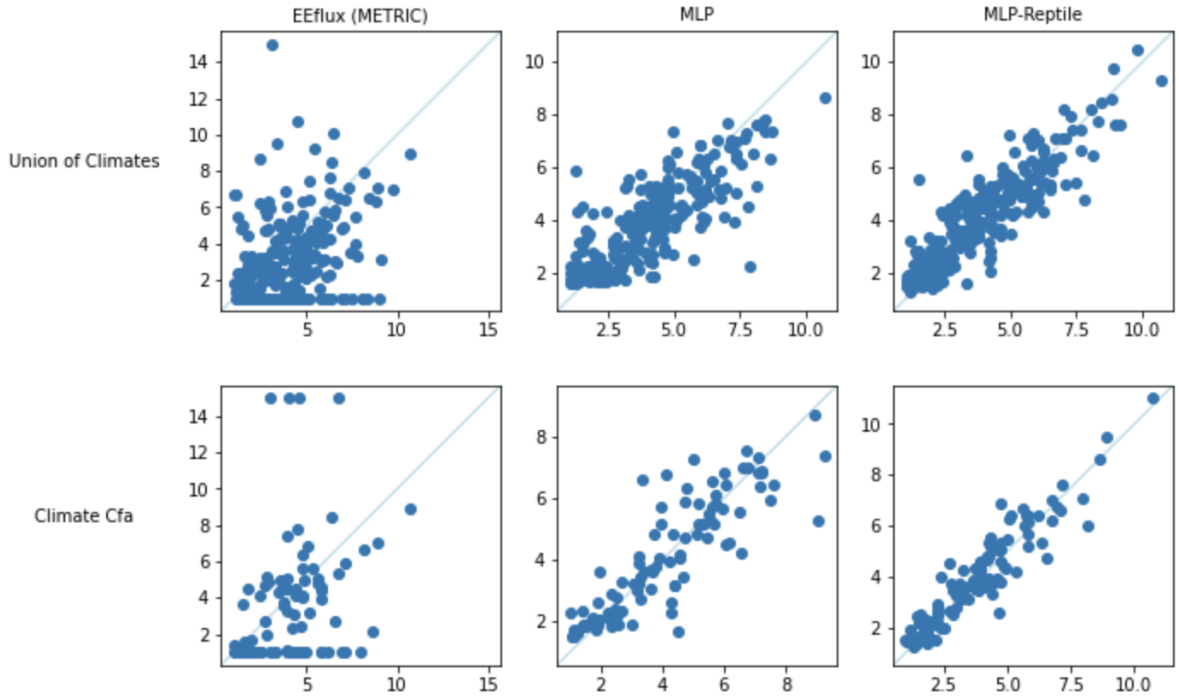


Figure 13.7: Experiment A Scatter Plot for Random Sampling

Figure [13.7] illustrates on the x-axis the Real ET (mm) and on the y-axis the Predicted ET (mm). It shows the testing data set for MLP (base model), MLP-Reptile (best model), and EEflux (Metric) model across the best climates and the union of climates ordered according to their performance. It is noticed that the MLP-Reptile shows a better diagonal fit than the MLP model and EEFlux (METRIC) model across Climate Cfa and Union of Climates. The points in the

EEflux (METRIC) model were scattered and not centered around the bisector. MLP-Reptile trained on Climate Cfa yielded a better concentration around the bisector in comparison to the union of climates.

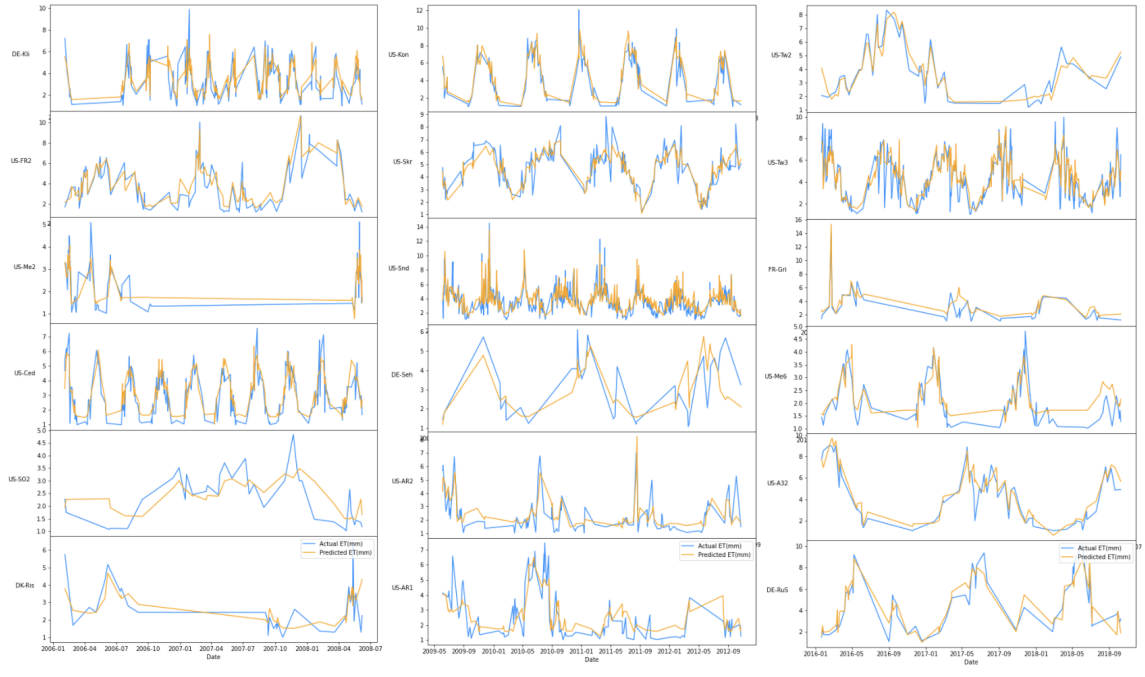


Figure 13.8: Line plots for MLP-Reptile across all Sites

Figure [13.8] shows Real ET versus Predicted ET for all the sites for our testing data for the MLP-Reptile model.

The x-axis represents the date in years and the y-axis represents the Real ET versus the Predicted ET for each site.

As shown in **Figure [13.8]**, Predicted ET by MLP-Reptile model tracks excellently the Real ET in almost all of the years for all the sites.

We further zoom into one of these sites (site US-Kon)

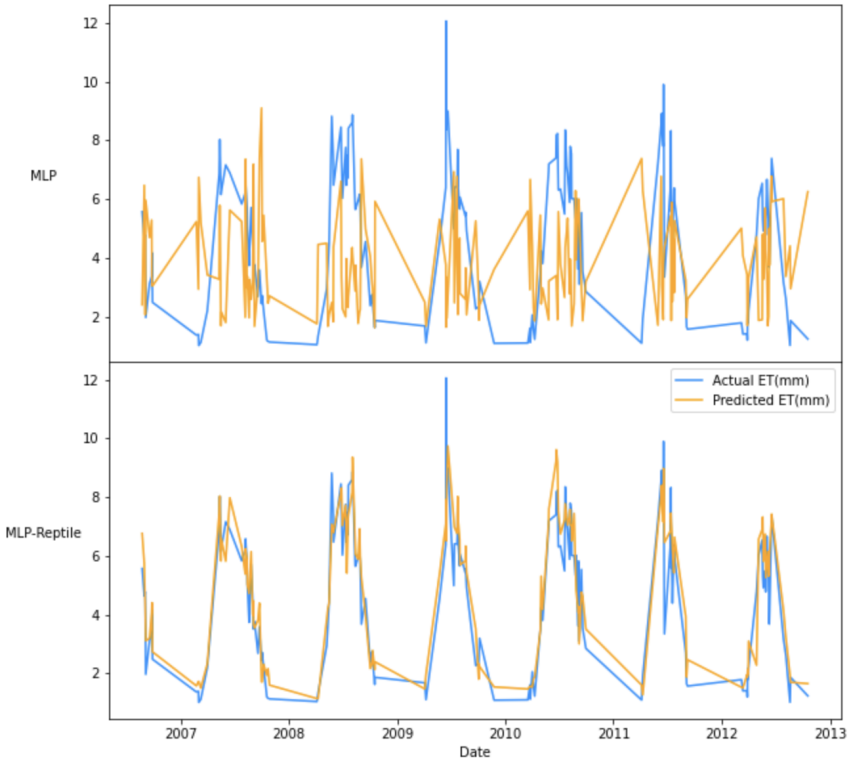


Figure 13.9: Line plots for MLP and MLP-Reptile for US-Kon

Figure [13.9] shows Real ET versus Predicted ET as a function of daily data per years (2007 - 2013) for site US-Kon for our testing data across two models: MLP and MLP-Reptile (top to bottom respectively) for the sampling randomly approach.

The top sub-figure shows the Real ET (blue line) versus the Predicted ET (orange line) values across the years for the MLP model. It is noticed that the Predicted ET poorly tracks the Real ET in almost all the date intervals.

The bottom sub-figure shows that the Predicted ET by MLP-Reptile model tracks excellently the Real ET in almost all of the years outperforming the MLP model.

We have further conducted a residual analysis study for the union of clusters and the best performing clusters (Cfa, and Cwa) by studying the performance of Experiment B, Experiment C, and Experiment D across different models as shown in **Figure [13.10]**

Figure [13.10] shows the experimental results across the best climate clusters and their union. We contrast the performance of the MLP (base model) and the MLP-Reptile (best model) in minimizing the residual biases as mentioned in

	Clusters	Union of Clusters			Climate Cfa			Climate Cwa		
Models	Average ET	3.74581288			4.00774363			4.51677034		
	Residuals	Experiment B	Experiment C	Experiment D	Experiment B	Experiment C	Experiment D	Experiment B	Experiment C	Experiment D
	F2	0.88	0.96	0.88	0.94	0.97	0.94	0.00001	0.00001	0.00001
	Recall	0.87	0.96	0.87	0.94	0.97	0.94	0.00001	0.00001	0.00001
	R2	0.85	0.87	0.85	0.96	0.95	0.96	0.97	0.96	0.97
	RMSE	0.33	0.91	0.33	0.17	0.72	0.17	0.09	0.47	0.09
	MAE	0.17	0.69	0.17	0.11	0.56	0.11	0.06	0.39	0.06
	Accuracy	81.32	NA	NA	84.14	NA	NA	90.32	0.38	21.12
	NMI	1	1	1	1	1	1	1	1	1
	Training Time (seconds)	NA	NA	NA	NA	NA	NA	NA	NA	NA
MLP-Reptile	Testing Time (seconds)	NA	NA	NA	NA	NA	NA	NA	NA	NA
	F2	0.90409877	0.95434995	0.90409877	0.82050605	0.95123475	0.82050605	0.8794761207	0.876168284	0.8794761207
	Recall	0.9075685	0.96087003	0.9075685	0.82252352	0.95476534	0.82252352	0.8778934055	0.90680395	0.8778934055
	R2	0.89104851	0.79241899	0.89104851	0.83533646	0.90550819	0.83533646	0.82581324	0.838308139	0.82581324
	RMSE	0.25743452	1.17444166	0.25743452	0.52665946	1.02763797	0.52665946	0.2335694667	1.128385327	0.2335694667
	MAE	0.1592884	0.86227765	0.1592884	0.20473133	0.71031845	0.20473133	0.152916216	0.9789949276	0.152916216
	Accuracy	74.6067405	NA	NA	80.2056478	0.78499978	17.5142884	74.87414534	NA	NA
	NMI	1	1	1	1	1	1	1	1	1
	Training Time (seconds)	NA	NA	NA	NA	NA	NA	NA	NA	NA
MLP	Testing Time (seconds)	NA	NA	NA	NA	NA	NA	NA	NA	NA
	F2	-	-	-	-	-	-	-	-	-
	Recall	-	-	-	-	-	-	-	-	-
	R2	-	-	-	-	-	-	-	-	-
	RMSE	-	-	-	-	-	-	-	-	-
	MAE	-	-	-	-	-	-	-	-	-
	Accuracy	-	-	-	-	-	-	-	-	-
	NMI	-	-	-	-	-	-	-	-	-
	Training Time (seconds)	-	-	-	-	-	-	-	-	-
	Testing Time (seconds)	-	-	-	-	-	-	-	-	-
Stat	Data set Size	10,911	10,911	10,911	3,645	3,645	3,645	674	674	674

Figure 13.10: Residual Analysis for Random Sampling

Experiments B, C, and D. We note the following:

- MLP-Reptile outperforms all models in minimizing the absolute bias across the union of clusters, producing an R2 score of 0.87, and MAE of 0.69
- MLP-Reptile outperforms all others in minimizing the proportional bias across the union of clusters, producing an R2 score of 0.85, MAE of 0.17, and Accuracy of 81.32%
- MLP-Reptile outperforms all others in minimizing the combined bias across the union of clusters, producing an R2 score of 0.85, and MAE of 0.17
- Experiment B was the best in terms of all the reported metrics compared to Experiment C and Experiment D. Thus, minimizing the proportional bias was the most successful residual analysis

In **Figure** [**13.11**], the columns represents Experiments B, C, and D respectively and the row represents the two variables as been defined for each experiment in **Chapter** [**12**] for MLP-Reptile Model as being our best model for the union of climates. We note that Experiments B and D show very comparable visualizations having the points scattered around the bisector with a bit more distribution as opposed to Experiment C. Experiment C show a better diagonal than the others. However, quantitative results as shown in **Figure** [**13.10**] indicate that

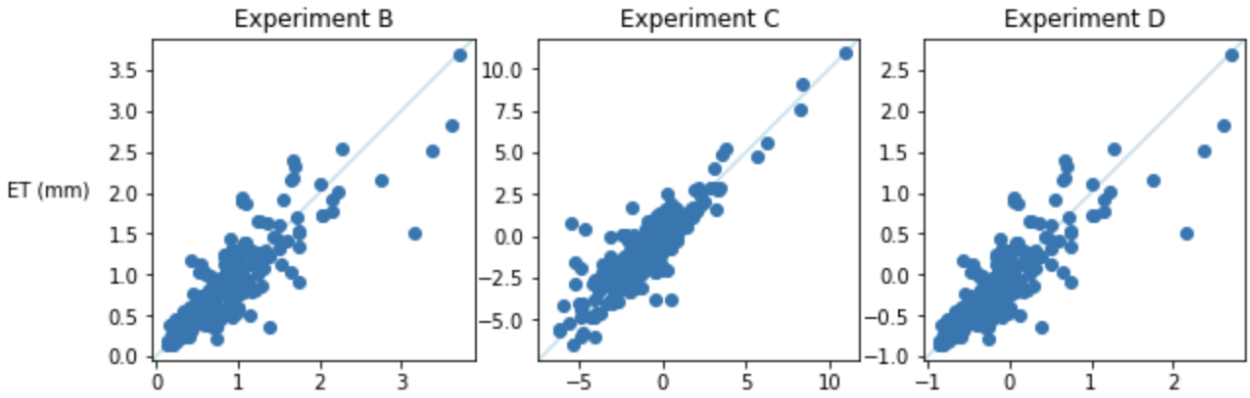


Figure 13.11: Residual Analysis Scatter Plots for Random Sampling

MLP-Reptile was the best point-wise model in minimizing the proportional bias, which is represented by Experiment B.

N.B: AutoML results are not reported as earlier since we could not extract the testing data set for us to compute the above metrics.

Takeaway Message: MLP-Reptile was the best point-wise model to predict Real ET. MLP-Reptile beats MLP in R2 by 31.3% to 43.63% and in RMSE by 24.3% to 48.9%. MLP-Reptile also beats AutoML in R2 by 8.2% to 8.6% and in RMSE by 10.8% to 48.9%. MLP-Reptile performs best in climates Cfa and Cwa (Humid Subtropical - mild with a dry season and hot summer), which is considered suitable for farmers who would need to know irrigation factor values the most in these types of climates, rather than a climate with rich irrigation and moist seasons. MLP-Reptile trained on climates (Cfa, and Cwa) beat the union of climates in R2 by 7.05% to 10.2%, in Accuracy by 5.3% to 14.8%, and in Recall by 0.72% to 1.94%. Also **Figure [13.7]** confirms our quantitative observations present in **Figure [13.6]**. Moreover, MLP-Reptile was the best model to minimize the proportional bias i.e Experiment B. MLP-Reptile beats MLP model in R2 by 6.09% -8.98% and in RMSE by 36.5% - 60.86%.

13.5.3 Sampling Tasks By Climate

We have conducted Experiment A across MLP-Reptile, MLP, EEflux (METRIC), AutoML, and Stat on our daily data (union of clusters) and on different clusters where each experiment represents training on 5 climates and testing on a new unseen climate/cluster. The best feature selection scenario yielding the best Accuracy, Recall, and the least error metrics was “Scenario C”. We will show the

results for that scenario and all other scenarios will be presented in the appendix.

We have performed six experiments. The number of episodes for each cluster is summarized in **Table** [**13.1**] for the sampling by climate approach. Experiments are defined as such:

- Cluster Other: We train on 1500 episodes on climates Cfa, Csa, Csb, Cwa, and Dsa. Each climate is being trained on 300 episodes and we further tested on climate Other.
- Cluster Csb: We train on 1500 episodes on climates Cfa, Csa, Dsa, Cwa, and Other. Each climate is being trained on 300 episodes and we further tested on climate Csb.
- Cluster Dsa: We train on 1500 episodes on climates Cfa, Csa, Csb, Cwa, and Other. Each climate is being trained on 300 episodes and we further tested on climate Dsa.
- Cluster Csa: We train on 1500 episodes on climates Cfa, Cwa, Dsa, Csb, and Other. Each climate is being trained on 300 episodes and we further tested on climate Csa.
- Union of Clusters: We randomly train on 70% of the data and test on 30% of the data as we performed when sampling randomly.
- Cluster Cfa: We train on 1500 episodes on climates Csa, Cwa, Dsa, Csb, and Other. Each climate is being trained on 300 episodes and we further tested on climate Cfa.
- Cluster Cwa: We train on 1500 episodes on climates Cfa, Csa, Dsa, Csb, and Other. Each climate is being trained on 300 episodes and we further tested on climate Cwa.

	Clusters	Cluster Other	Cluster Csb	Cluster Dsa	Cluster Csa	Cluster Cwa	Cluster Cfa	Union of Clusters
Models	**Average ET**	3.50686273	2.4071607	2.6196219	4.00808472	4.51677034	4.00774363	3.74581288
	Train Size	1167	431	565	2457	472	2552	7638
	Test Size	499	184	242	1,052	202	1,093	3,273
	Recall	0.0001	0.0001	0.0001	0.0001	0.0001	0.73	0.78
	F2	0.0001	0.0001	0.0001	0.0001	0.0001	0.72	0.78
	R2	-30.577819	-1.4356181	-0.888912	-0.4214216	-0.4214216	-1.8242985	-0.9572085
	RMSE	3.82441468	1.37960202	2.51340802	2.35727511	2.35727511	3.33013862	2.79025128
EEflux (METRIC)	**MAE**	2.38679044	1.2173055	2.02203571	1.54465131	1.54465131	2.39641085	2.1026953
	Accuracy	NA	41.8801079	42.6522932	69.4002766	69.4002766	33.7737912	39.1700276
	NMI	0.98412698	1	0.86805742	0.98870571	0.98870571	0.82271656	0.90721809
	Training Time (seconds)	NA	NA	NA	NA	NA	NA	NA
	Testing Time (seconds)	NA	NA	NA	NA	NA	NA	NA
	Recall	0.95836498	0.00001	0.9968438	0.95078851	0.97061932	0.97244863	0.96005225
	F2	0.95317854	0.00001	0.98343192	0.95194721	0.96932869	0.96443131	0.96157961
	R2	0.51502544	0.64649444	0.66991301	0.64660694	0.70194802	0.74999103	0.79383871
	RMSE	1.28065653	0.52061519	0.78535687	1.18139397	0.81465905	1.0435972	0.9088138
MLP-Reptile	**MAE**	0.97235766	0.42287481	0.66053179	0.86498617	0.62125474	0.80548712	0.67232353
	Accuracy	58.9122087	78.0025861	65.3091185	70.7481218	82.8338769	70.7639216	76.6164768
	NMI	0.99949038	0.97077763	0.97500197	0.99964107	0.99935315	0.98730802	0.99512033
	Training Time (seconds)	532.017945	446.64288	490.45944	478.31976	514.83822	465.90543	946.89702
	Testing Time (seconds)	3.90707994	3.15307021	3.31060839	3.65248013	3.88436794	3.30065608	97.1628394
	Recall	0.88680246	0.00001	0.89798946	0.92673028	0.9497509582	0.9297036775	0.9259478
	F2	0.9110401	0.00001	0.00004	0.93378058	0.00004	0.9406024293	0.9378146
	R2	0.51188491	0.51104046	0.40678114	0.60148922	0.5571167	0.6795279101	0.65130298
	RMSE	1.40731786	0.65575748	1.14578606	1.22366767	0.98225045	1.195869171	1.17380524
MLP	**MAE**	1.00921214	0.52224957	0.84170251	0.87698342	0.75835168	0.8242660495	0.82042826
	Accuracy	63.5327184	74.4137471	63.4869533	74.3498917	76.6873862	74.58439234	74.328515
	NMI	1	1	1	1	1	1	1
	Training Time (seconds)	131.949494	285.46254	168.297828	156.614623	425.939502	190.669224	260.793119
	Testing Time (seconds)	0.16196299	1.27572989	0.9590857	0.55923653	0.68710041	0.58198786	1.1848011
	Recall	-	-	-	-	-	-	-
	F2	-	-	-	-	-	-	-
	R2	-	-	-	-	-	-	-
	RMSE	-	-	-	-	-	-	-
Stat	**MAE**	-	-	-	-	-	-	-
	Accuracy	-	-	-	-	-	-	-
	NMI	-	-	-	-	-	-	-
	Training Time (seconds)	-	-	-	-	-	-	-
	Testing Time (seconds)	-	-	-	-	-	-	-
	Recall	-	-	-	-	-	-	-
	F2	-	-	-	-	-	NA	NA
	R2	-	-	-	-	-	0.801	0.736
	RMSE	-	-	-	-	-	0.988	1.012
Auto ML	**MAE**	-	-	-	-	-	0.706	0.71
	Accuracy	-	-	-	-	-	77.37	75.95
	NMI	-	-	-	-	-	NA	NA
	Training Time (seconds)	-	-	-	-	-	7200	7200
	Testing Time (seconds)	-	-	-	-	-	NA	NA
	Data Size	1666	615	807	3,509	674	3,645	10,911

Figure 13.12: Experiments for Climate Sampling

Figure [13.12] portrays the experimental results across different climate clusters and the union of clusters. We compare the performance of our four models: EEflux (METRIC), MLP (base model), MLP-Reptile (best model), and Auto ML. We note that MLP-Reptile is performing the best among all others in predicting real ET across all cluster variations as been noted in **Figure [13.6]**. However, when experimenting with climate Csb, the results have improved substantially with an R2=0.52 to an R2=0.64. This often emphasizes the power of transfer learning from other pre-trained climates as this experiment have learned from training on climates Cfa, Cwa, Csa, etc..

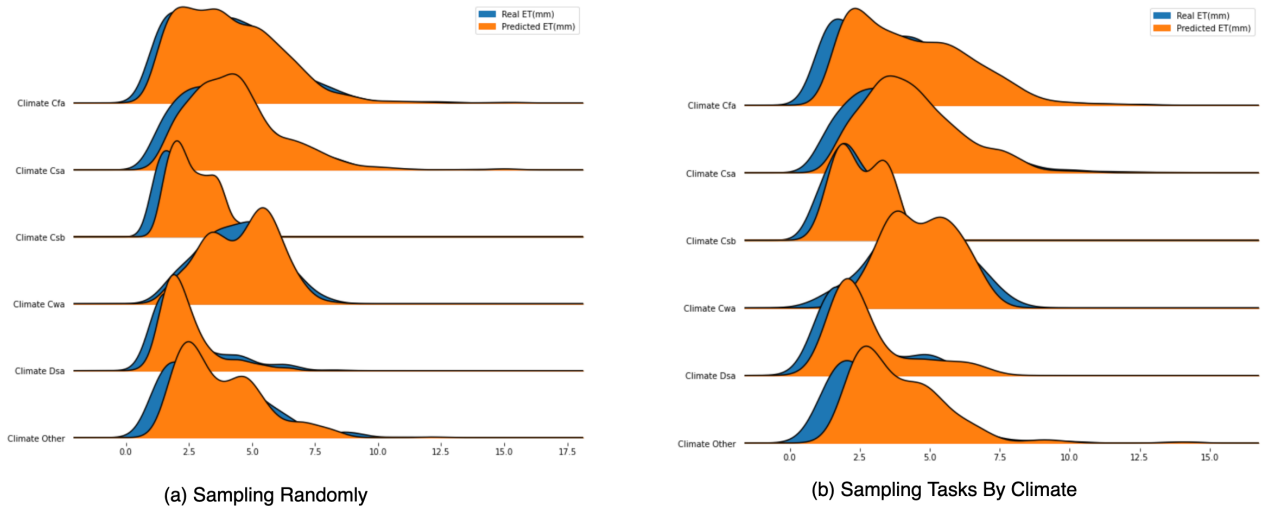


Figure 13.13: Sampling Tasks Randomly vs Sampling Tasks By Climate

In **Figure [13.13]**, the x-axis shows ET (mm) where the blue color represents the Real ET and the orange color represents the Predicted ET(mm) and the y-axis shows the density plot for each climate. We plot Real ET versus Predicted ET across all the available climates (Cfa, Csa, Csb, Cwa, Dsa, and Other) and we compare the plots for when we sample the tasks randomly or sample tasks by climate. We note that climate Cfa, which is proven to be the best performing in **Figure [13.10]** and **Figure [13.12]**, offers the best trace for Real versus Predicted ET (mm). Climate Csa and Cwa also show comparable performance, unlike Dsa, Csb, and Other.

Takeaway Message: MLP-Reptile was the best point-wise model to predict

Real ET. Randomly sampling our data seems to be a better candidate than sampling by climates for all the climate experiments excluding climate Csb where R2 has improved substantially with an R2=0.52 to an R2=0.64.

In this approach, we do not have a cluster achieving better than the union of clusters. The residual analysis will only be performed for the union of clusters which has the same results as mentioned in **Figure [13.10]**.

13.5.4 Sampling Tasks By Season

After observing model performance on climate clusters, we now aim to study model performance across different seasons. We identify three seasons: Summer, Winter, and Spring. We note that clustering by TA using Kmeans, where $k = 3$ was the most successful in showing distinct seasons per clusters (more details are explained in **Chapter [7]**). We have conducted Experiment A across MLP-Reptile, MLP, EEflux (METRIC), AutoML, and Stat on our daily data (union of clusters) and on different seasons that are shuffled randomly with a split ratio of 70/30. The best feature selection scenario yielding the best Accuracy, Recall, and the least error metrics was "Scenario A". We will show the results for that scenario and all other scenarios will be presented in the appendix.

We have performed three experiments where each experiment represents training and testing in the same season. The number of episodes for each season is summarized in **Table [13.2]** for the sampling tasks by season approach. Experiments were defined as such:

- Spring: We randomly train on 3000 episodes using a 70/30 split on the spring season data set that are clustered by TA using Kmeans, $k = 3$ on the first cluster
- Winter: We randomly train on 3000 episodes using a 70/30 split on the winter season data set that are clustered by TA using Kmeans, $k = 3$ on the second cluster
- Union of Clusters: We randomly train on 3000 episodes on 70% of the data and test on 30% of the data for all the seasons
- Summer: We randomly train on 3000 episodes using a 70/30 split on the summer season data set that are clustered by TA using Kmeans, $k = 3$ on the third cluster

	Clusters	Spring	Winter	Union of Clusters	Summer
Models	**Average ET**	3.60	2.31	3.75	4.95
	Train Size	3510	2173	9535	3853
	Test Size	1,504	930	4,086	1,651
	Recall	0.0001	0.61	0.7380189	0.0001
	F2	0.0001	0.62	-1.0526854	0.0001
	R2	-1.60	-2.65	-1.0526854	-1.24
	RMSE	2.59	2.72	2.90773858	3.19
EEflux (METRIC)	**MAE**	2.06	1.82	2.20007924	2.47
	Accuracy	51.06	20.42	41.2920661	46.22
	NMI	0.93	0.89	1.00	0.94
	Training Time (seconds)	NA	NA	NA	NA
	Testing Time (seconds)	NA	NA	NA	NA
	Recall	0.98010285	0.98848097	0.96005225	0.96857497
	F2	0.97435973	0.98155799	0.96157961	0.96950832
	R2	0.73425867	0.78356457	0.79383871	0.8188632
	RMSE	0.83562006	0.59280902	0.9088138	0.90699729
MLP-Reptile	**MAE**	0.65292369	0.45399813	0.67232353	0.67363146
	Accuracy	75.8003462	76.4829892	76.6164768	82.102536
	NMI	0.98439695	0.98051135	0.99512033	0.99365577
	Training Time (seconds)	875.09322	875.09322	946.89702	890.09322
	Testing Time (seconds)	36.9717534	30.0078702	97.1628394	32.3838
	Recall	0.94	0.85	0.93551997	0.93
	F2	0.94	0.00005	0.94241968	0.94
	R2	0.58	0.27	0.6655619	0.52
	RMSE	1.41	0.96	1.11807771	1.12
MLP	**MAE**	1.02	0.69	0.81582267	0.82
	Accuracy	73.54	64.85	73.3617402	71.14
	NMI	1.00	1.00	1.00	1.00
	Training Time (seconds)	189.84	85.08	280.67	159.47
	Testing Time (seconds)	0.13	0.12	0.22	0.15
	Recall	-	-	-	-
	F2	-	-	-	-
	R2	-	-	-	-
	RMSE	-	-	-	-
Stat	**MAE**	-	-	-	-
	Accuracy	-	-	-	-
	NMI	-	-	-	-
	Training Time (seconds)	-	-	-	-
	Testing Time (seconds)	-	-	-	-
	Recall	-	-	-	-
	F2	-	-	NA	-
	R2	-	-	0.736	-
	RMSE	-	-	1.012	-
Auto ML	**MAE**	-	-	0.71	-
	Accuracy	-	-	75.95	-
	NMI	-	-	NA	-
	Training Time (seconds)	-	-	7200	-
	Testing Time (seconds)	-	-	NA	-
	Data Size	5,014	3,103	13,621	5,504

Figure 13.14: Experiments for Seasonality Sampling

Figure [13.14] portrays the experimental results across different season clusters and the union of clusters. We compare the performance of our models: EEflux (METRIC), MLP (base model), and MLP-Reptile (best model). We note that MLP-Reptile is performing the best among all others in predicting Real ET across all cluster variations, producing an R2 score of 0.81, an RMSE of 0.90, and an Accuracy of 82% for the summer season. All models performed well on all the clusters/season but with showing the best results for the summer season. The clusters are ordered from worst to best in terms of their results.

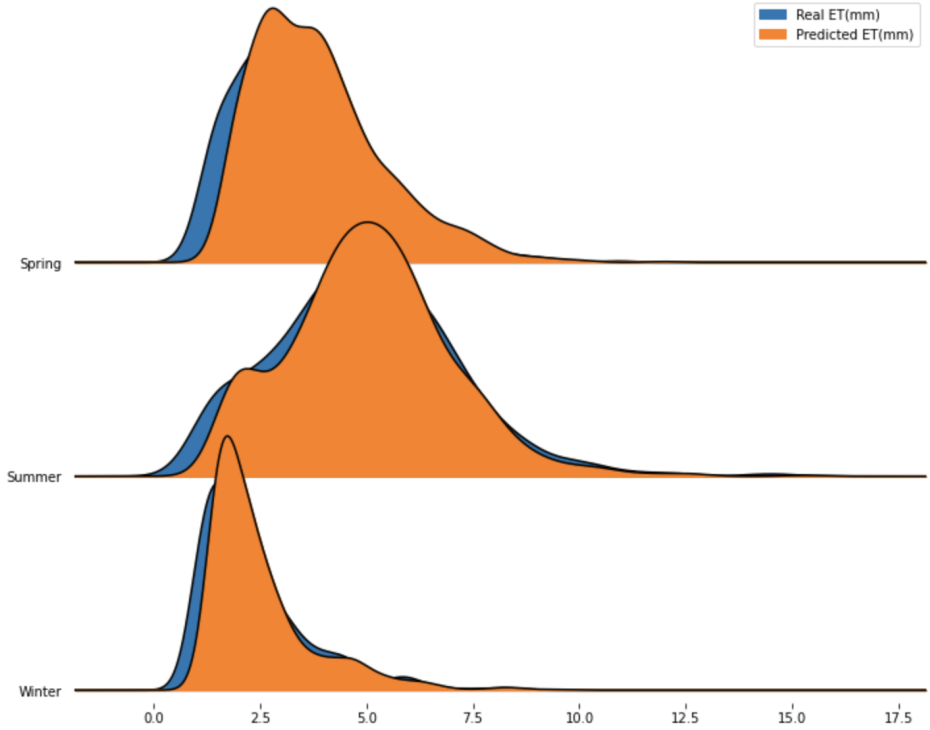


Figure 13.15: Density Plots for Sampling Tasks By Season

In **Figure [13.15]**, the x-axis shows ET (mm) where the blue color represents the Real ET and the orange color represents the Predicted ET(mm) and the y-axis shows the density plot for each season. We plot the testing results of each season after training our data using MLP-Reptile. The density plots for each figure show that the Predicted ET tracks well the Real ET for each season as the orange density plot is very close to the blue density plot.

Takeaway Message: MLP-Reptile was the best point-wise model to predict Real ET. MLP-Reptile beats MLP in R2 by 22.7% to 63% and in RMSE by 35.60283688% to 38.5%. MLP-Reptile trained on the summer season beat the union of clusters/seasons in R2 by 2.46%, in Accuracy by 6.58%, and in Recall

by 0.82%. The best performing season was the summer season i.e cluster 0.

We have further conducted a residual analysis study for the union of clusters and the best performing season (summer) by studying the performance of Experiment B, Experiment C, and Experiment D across different models as shown in **Figure [13.16]**

	Clusters	Union of Clusters			Summer		
Models	Average ET	3.74581288			4.95404614		
	Residuals	Experiment B	Experiment C	Experiment D	Experiment B	Experiment C	Experiment D
	F2	0.88	0.95	0.88	0.85	0.96	0.85
	Recall	0.87	0.94	0.87	0.86	0.97	0.86
	R2	0.83	0.80	0.83	0.85	0.89	0.85
	RMSE	0.36	1.21	0.36	0.35	0.89	0.35
	MAE	0.17	0.89	0.17	0.13	0.68	0.13
	Accuracy	78.30	NA	NA	85.77	NA	NA
	NMI	1	1	1	1	1	1
	Training Time (seconds)	NA	NA	NA	NA	NA	NA
MLP-Reptile	Testing Time (seconds)	NA	NA	NA	NA	NA	NA
	F2	0.90409877	0.95434995	0.90409877	0.8216872112	0.9198795306	0.8216872112
	Recall	0.9075685	0.96087003	0.9075685	0.8151052758	0.9268766942	0.8151052758
	R2	0.89104851	0.79241899	0.89104851	0.7810277976	0.8002154798	0.7810277976
	RMSE	0.25743452	1.17444166	0.25743452	0.3939926512	1.474056194	0.3939926512
	MAE	0.1592884	0.86227765	0.1592884	0.1999883284	1.124324691	0.1999883284
	Accuracy	74.6067405	NA	NA	77.81998631	NA	NA
	NMI	1	1	1	1	1	1
	Training Time (seconds)	NA	NA	NA	NA	NA	NA
MLP	Testing Time (seconds)	NA	NA	NA	NA	NA	NA
	F2	-	-	-	-	-	-
	Recall	-	-	-	-	-	-
	R2	-	-	-	-	-	-
	RMSE	-	-	-	-	-	-
	MAE	-	-	-	-	-	-
	Accuracy	-	-	-	-	-	-
	NMI	-	-	-	-	-	-
	Training Time (seconds)	-	-	-	-	-	-
	Testing Time (seconds)	-	-	-	-	-	-
Stat	Data set Size	10,911	10,911	10,911	5,504	5,504	5,504

Figure 13.16: Residual Analysis for Seasonality Sampling

Figure [13.16] shows the experimental results across the best season cluster and its union. We contrast the performance of the MLP (base model) and the MLP-Reptile (best model) in minimizing the residual biases as mentioned in Experiments B, C, and D. We note the following:

- MLP-Reptile outperforms all models in minimizing the absolute bias across the union of clusters, producing an R2 score of 0.80, and MAE of 0.89
- MLP-Reptile outperforms all models in minimizing the proportional bias across the union of clusters, producing an R2 score of 0.83, MAE of 0.17, and Accuracy of 78.30%
- MLP-Reptile outperforms all others in minimizing the combined bias across the union of clusters, producing an R2 score of 0.83, and MAE of 0.17

- Experiment B was the best in terms of all the reported metrics compared to Experiment C and Experiment D. Thus, minimizing the proportional bias was the most successful residual analysis

13.6 Takeaway Message

Sampling randomly showed the best results for all the climates as opposed to sampling by climates except for climate Csb. Sampling randomly showed an improvement from MLP to MLP-Reptile for climate Cwa by 57.14% in R2 and 48.9% in RMSE.

Moreover, when sampling by seasons results was also well-performing, yielding the highest accuracy and R2 for the summer season with R2=0.81 and Accuracy=82% even better than the union of clusters by 2.46% in R2, 6.69% in Accuracy, and 0.83% in Recall as indicated in **Figure [13.14]**. Also, for all the sampling approaches, MLP-Reptile was the best model to minimize the proportional bias i.e Experiment B. MLP-Reptile beats MLP model in R2 by 6.09% - 8.98% and in RMSE by 36.5% - 60.86% for when sampling randomly.

13.7 Models' Stability and Performance

For all our experiments, we have evaluated our testing data set on 10-100 folds in order to yield consistent and stable results rather than random ones. We will compare the performance of our models in terms of different metrics i.e (the most accurate, the least training time, and the one yielding the least bias and variance).

13.7.1 Most Accurate Model

When comparing our models in terms of accuracy, it is noted that MLP-Reptile is the best point-wise model with an accuracy of 76.6% as opposed to the MLP model yielding an accuracy of 74.33%.

13.7.2 Most Precise Model

When comparing our models in terms of utility-based metrics, it is noted that MLP-Reptile is the best point-wise model with a recall of 0.96, and precision of 0.97 as opposed to the MLP model yielding a recall of 0.93, and precision of 0.99. Thus the MLP-Reptile model is a better candidate in capturing correctly rare from non-rare values as it has a higher recall than an MLP model.

13.7.3 Least Training Time Model

When comparing the training time of our models, MLP took 260 seconds for learning with 500 epochs, however, MLP-Reptile took much more time optimizing the training process with 1,665 seconds with 3000 epochs/episodes. MLP-Reptile requires more time to converge to a better solution, however, it can learn with only a few-shot examples.

13.7.4 Learning Experience

We have computed validation scores, validation standard deviation, and evaluated our model on a shuffled testing data set for each fold. In addition to that, we have outputted a learning curve which represents the mean squared error loss versus the number of epochs between the training and validation data sets.

- For MLP Model: We have computed validation scores and plotted the learning curve.
  - MLP yields an accuracy of 74.33% for the testing data set, and an accuracy of 73.34% for the validation data set.
  - MLP yields an MSE of 1.38 mm for the testing data set, and an MSE of 1.37 mm for the validation data set.

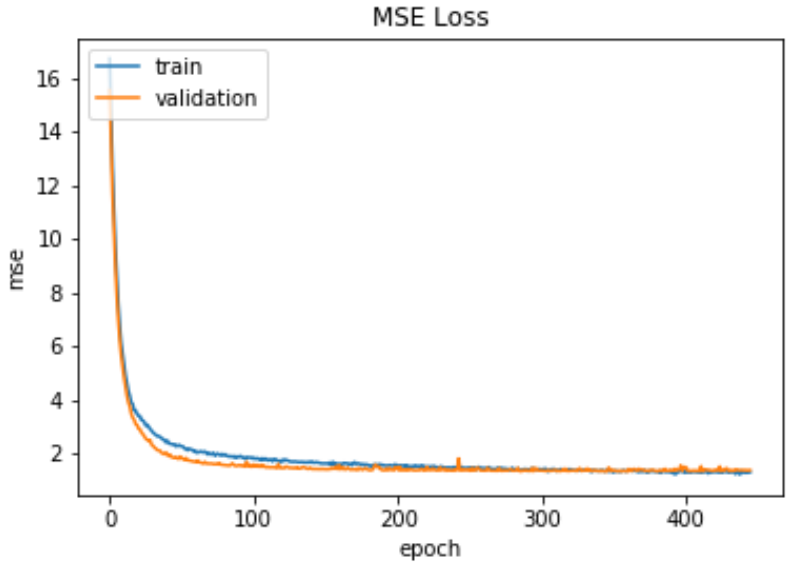


Figure 13.17: Learning Curve

As shown in **Figure [13.17]**, it is noted that as the number of epochs increase, the mean squared error loss decreases for both validation and training data sets. However, when the number of epochs is greater than 200, the

validation and training data set track each the one another well reaching a constant learning experience i.e the loss does not decrease/improve anymore. Thus our model is able to learn the output well and does not suffer from neither high bias or high variance.

- For MLP-Reptile Model: We have built our MLP-Reptile model on top of an MLP model and evaluated our testing data over 100 runs. For each run, we shuffled our testing data set and averaged the error metrics and accuracy measures. Thus resulting in stable results.

Chapter 14

Uncetainty Quantification

Quantifying uncertainty is becoming more relevant in machine learning and is often a necessity for clients. Especially when the implications of a wrong prediction are high, you need to know what the likelihood (distribution) of an individual prediction is. You're probably thinking about using Bayesian methods to quantify this. But these approaches have their downsides, too. For example, when dealing with large quantities of data or lots of parameters, it can be computationally costly. An example in real life where the implications of a wrong prediction are high is when using a medical diagnosis model. This model should not only take care about the accuracy but it should also quantify how certain the prediction is, if the uncertainty is too high, the doctor is notified to take this into account in his decision process. Another applicable example is self-driving cars that have not learned from sufficient data. In this case, if the car is unsure where there is a pedestrian on the road, we would expect it to let the driver take charge. Thus the need for quantifying uncertainty and further distinguishing between at least two types of uncertainty is often referred to as aleatoric and epistemic uncertainties.

14.1 Uncertainty Types

We need to distinguish between the type of uncertainty to calculate the uncertainty. There are plenty of different uncertainty types. We are concerned with capturing the difference between Aleatoric and Epistemic uncertainty. Aleatoric (alluded to as aleatory) uncertainty captures uncertainty in the data itself. Epistemic uncertainty is the uncertainty in the model (Kana, 2019).

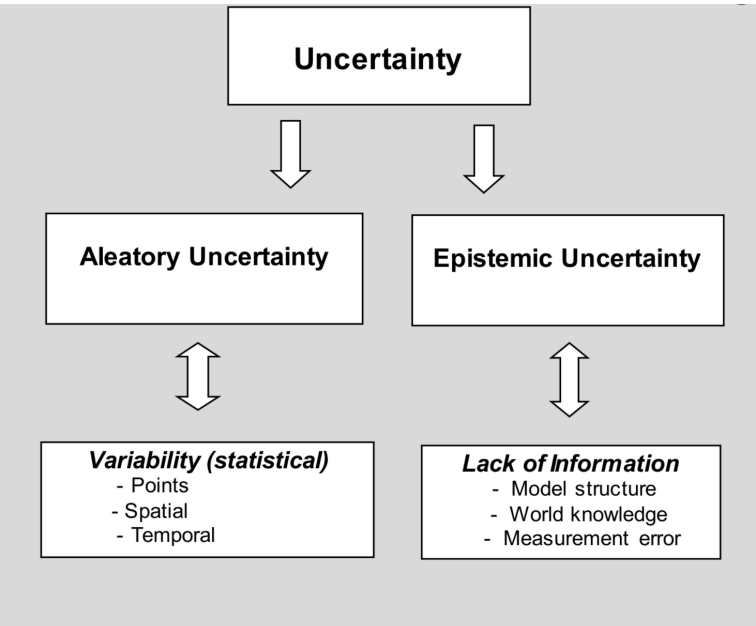


Figure 14.1: Uncertainty Types

14.1.1 Aleatoric Uncertainty

Aleatoric uncertainty (or statistical uncertainty) captures the uncertainty in the data generation process i.e. randomness in our data as being an irreducible aspect of the predictive variance. It is considered irreducible since it cannot be resolved by collecting more data since it is randomness from the data itself.
An aleatoric uncertainty is decomposed into homoscedastic uncertainty and heteroscedastic uncertainty. Homoscedastic uncertainty remains consistent regardless of the input value whereas the heteroscedastic uncertainty changes with having different input values.

An example of variability in data is as such: Let's consider we have house area to be our only input feature and the house price to be our output variable. It happens that for the same house area we have different house prices in the data set. This variance in the house price is referred to as aleatoric uncertainty.

Figure [14.2] represents a real linear process ($y = x$) that was sampled around $x = -2.5$ and $x = 2.5$.
A sensor malfunction presented noise in the left cloud which led to high aleatoric uncertainty. This uncertainty cannot be reduced by providing more measurements since the sensor keeps yielding errors around $x = -2.5$ by design (noise).

An aleatoric uncertainty can be estimated by having a deep neural network to output the parameters of a probability distribution $P(y|x)$ of the target.

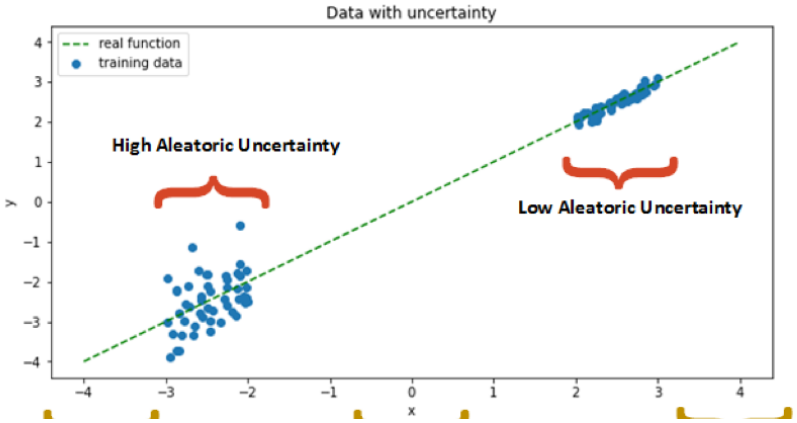


Figure 14.2: Aleatoric Uncertainty

Aleatoric uncertainty can be computed as such (Hullermeir & Waegeman, 2020):

$$aleatoric\_uncertainty = \sigma_{\epsilon}^2 \tag{14.1}$$

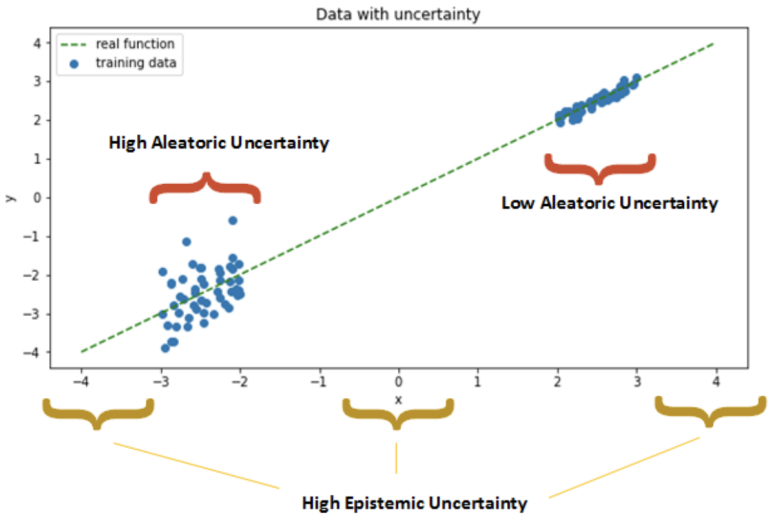
The variance of the error term σ_{ϵ}^2 corresponds to the aleatoric uncertainty. In our case the error term used is root mean squared error (RMSE). We use σ_{RMSE}^2 as σ_{ϵ}^2.
The unit used is (mm^2) which is the same as the unit of our target variable but squared i.e (ET).

We use deep ensemble as being our best model to quantify uncertainty. A deep ensemble consists of an ensemble of neural networks. Each neural network of this ensemble is tested on the testing data. Hence, per data point, we have an array of RMSEs RMSE pertaining to a neural network model in this ensemble. Hence, we can compute the variance of the RMSE per point. (Hullermeir Waegeman, 2020).

14.1.2 Epistemic Uncertainty

Epistemic uncertainty (or systematic uncertainty) is the uncertainty in the model itself describing what is not known to the model because of limited data and knowledge. An example of this uncertainty: I am uncertain about the number of people living in Lebanon, however, I can obtain this information by giving more evidence. Epistemic uncertainty can be reduced and explained given more enough training samples. It is also referred to as model uncertainty.

High epistemic ambiguity exists in regions where there is little to no observations for training. This ambiguity occurs because there are so many model parameters



An exhibit of the different kinds of uncertainty in a linear regression context (Image by Michel Kana).

Figure 14.3: Epistemic Uncertainty

that could describe the fundamental phenomena of the ground reality and this is often the case for the left i.e ($x = -4$ to $x = -3$), middle i.e ($x = -1$ to $x = 1$), and right i.e ($x = 3$ to $x = 4$) portions of our clouds as shown in **Figure [14.3]**. Here we are not sure which model parameters better describe the results. Given more data in this area, the uncertainty will be reduced. It is important to define such spaces in high-risk applications.

An epistemic uncertainty can be estimated by performing a Bayesian inference which aims to utilize the posterior distribution $P(\theta|D)$ that is obtained by applying Bayes' theorem. Bayes' theorem revolves around computing the probability of an event given some prior belief or knowledge as such:

$$P(\theta|D) = \frac{P(D|\theta) \times P(\theta)}{P(D)} \tag{14.2}$$

where:

- θ and D are any two independent events.
- $P(\theta)$: The prior distribution that represents our beliefs about the true value of the parameters.
- $P(D|\theta)$: The likelihood distribution that is dependent on $P(D)$.
- $P(D)$: The marginal distribution of the data or known as the evidence.

- $P(\theta|D)$: The posterior distribution that represents our belief about the parameter values after we have computed everything on the right-hand side taking the observed data into account.

Therefore we can compute the posterior distribution of our parameters using our prior beliefs updated with our likelihood using the Bayesian inference which is the process of identifying the population properties or a probability distribution from the given data using Bayes' Theorem. However, in practice, it is often considered hard to obtain samples from the true posterior.

Epistemic uncertainty can be computed as such (Hullermeir & Waegeman, 2020):

$$epistemic\_uncertainty = {\sigma^2}_{PDF} \tag{14.3}$$

where σ^2_{PDF} is the variance of the posterior distribution or probability density function pertaining to the probability distribution predicted by the probabilistic model.
The unit used is (mm^2) which is the same as the unit of our target variable but squared i.e (ET).

14.1.3 Total Uncertainty

Total uncertainty is computed after having calculated the epistemic and aleatoric uncertainties.

$$total\_uncertainty = epistemic\_uncertainty + aleatoric\_uncertainty \tag{14.4}$$

Chapter 15

Probabilistic and Uncertainty Modeling

The previous point-wise models often predict one output value on every set of input features. However, point estimates are not always a good choice since they are prone to be erroneous and are limited to presenting one option rather than an allowed range. Hence, we introduced models outputting a probability distribution over an output space and we have incorporated uncertainty quantification in each model's predictions as mentioned in **Chapter [14]**. It is important to quantify uncertainty since a machine learning model is considered of no use if having low error metrics, high accuracy, but often uncertain. We will be experimenting with several probabilistic and uncertainty models as such:

- Mixed Density Network
- Monte-Carlo Dropout (MC Dropout)
- Deep Ensemble

15.1 Mixed Density Networks

Mixed Density Networks (MDN), a variant of the neural network, was developed by (Bishop, 1994). It maps multiple outputs to a single input. MDN is similar to a standard neural network but only differs in having the final layer mapped to a mixture of distributions. It predicts a class of probability distribution referred to as a Mixture Gaussian distributions for the output.
For each input x, we will predict a probability distribution function (pdf) that is a probability-weighted sum of multiple Gaussian probability distributions with different means and standard deviations.

A sampling-free uncertainty-aware estimation method is proposed by (Sungjoon Choi

& Oh, 2017) which utilizes a mixture density network for modeling complex distributions. This algorithm uses single stochastic forward paths to drop out an approach for uncertainty acquisition. Uncertainty is further decomposed into two categories: an explained and unexplained variances corresponding to epistemic and aleatoric uncertainties. Learning from the demonstration (Lfd) method is proposed where it is used to switch to a rule-based approach when an MDN captures an explained variance when training an aggressive controller in a simulated environment. Lfd was applied to a real-world driving data set from the US Highway and it outperformed other methods in terms of safety in the sense that it had captured the explained variance on the out-of-distribution inputs.

In addition to what is proposed in (Sungjoon Choi & Oh, 2017), we have further tuned the model's hyper-parameters and studied different input combinations and their impact on our model's performance in terms of utility-based regression metrics and uncertainty.

15.1.1 Architecture

The probability distribution is formalized as such:

$$P(Y = y|X = x) = \sum_{i}^{K} \Pi_i(x)\phi(y, \mu_i(x), \sigma_i(x)) \tag{15.1}$$

Equation 15.1: Taken from (otoro.net, 2015)

- K: Total number of Gaussians or components in a mixture.
- Π_i: It acts as multiplier or weight, for every i Gaussian. All weights must sum to one: $\sum_i^K \Pi_i = 1$.
- μ_i: It is the mean for the i. Gaussian.
- σ_i: It is the standard deviation for the i Gaussian. All of σ_i are positive.
- ϕ is the Gaussian function for a given mean and standard deviation. $\phi(y, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{1}{2}(\frac{y-\mu}{\sigma})^2)$.

1. Our model will consist of several hidden layers, each is formalized as: $z_h = \text{activationFunction}(W_{in}x + b_{in})$. The activationFunction could be a tanh, relu, etc...
2. The output layer will consist of our three parameters Π, σ, μ
   $z_\Pi = W_\Pi z_h + b_\Pi$
   $z_\sigma = W_\sigma z_h + b_\sigma$
   $z_\mu = W_\mu z_h + b_\mu$

3. For our loss function, we aim in minimizing the negative log-likelihood of the distribution or maximizing the logarithm of the likelihood of the output distribution.
   $loss(Y = y|X = x) = -\log\left(\sum_i^K \Pi_i(x)\phi(y, \mu_i(x), \sigma_i(x))\right)$

4. We will then use each in order to determine the parameters of the mixture of Gaussian distributions that can be used further.
   $\Pi_i = \frac{\exp(z_{\Pi_i})}{\sum_j^K \exp(z_{\Pi_j})}$
   $\sigma_i = \exp(z_{\sigma_i})$
   $\mu_i = z_{\mu_i}$

15.1.2 Hyper-parameters

We experimented with several experiments by varying some of the following hyper-parameters:

- Number of Epochs: The number of iterations the training data set is being shown to the neural network model while training.
- Dropout: A regularization technique that revolves around randomly dropping neurons so that their weights will not be updated resulting in improving generalizing and decreasing a model's ability to over-fit. It ranges from 0 to 0.5.
- Learning Rate: A tuning parameter that determines the step size at each iteration while moving toward a minimum of a loss function
- Number of layers: The number of hidden layers in our model between the input and output layers.
- Hidden 1: The number of hidden neurons for the first layer.
- Hidden 2: The number of hidden neurons for the second layer.
- Activation: A function that is attached to each neuron in the network and decides if a neuron should be fired or not based on its input if this input happens to be relevant to the model's prediction. It also aids in mapping the output to a range between 0 and 1 or -1 and 1.
- Number of Mixtures: The number of Gaussian mixtures, this is being varied per experiment.
- Optimizer: A function that is defined to reduce the loss by updating the weights or learning rates.
- Distribution of the target: The distribution of our output variable.

15.1.3 Implementation

We have built an MDN network and employed hyper-parameters tuning and utility-based learning. The data set is divided into 70% training and 30% testing.

This module outputs testing scores, in addition to all necessary plots, figures, and data sets needed.

This module code is implemented using Tensorflow. Our code is inspired by the Github code which is available here, but we have further tuned the script files to match our implementation of providing dynamic hyper-parameters and dynamic data set processing with a real-world data set of 2 dimensions.

Our implementation is available here and divided into several files as such:

- colors.py: This python file contains a dictionary of colors per each uncertainty type and true versus predicted output.
- dataset_preparation.py: This python file contains the set up for our data set.
- distribution.py: This python file contains some exploratory analysis for checking our target distribution.
- mixture_model.py: This file contains the neural network model architecture which further initializes each input, hidden and an output layer of a neural network.
- mixture_training.py: This file contains the training process of a neural network.
- mixture_evaluation.py: This file contains the evaluation process of the neural network by training the neural network and evaluating it on the testing data set and further reporting the predictions with their uncertainty. It also includes the options to tune the hyper-parameters of the network.
- plotting.py: This file contains functions for plotting different uncertainties and plotting the real output and the predicted output and its variance.

We tuned the MDN hyper-parameters by utilizing a grid search and Utility-based regression. Each range of values for each hyper-parameter is chosen based on what is accepted in the literature.

- Number of Epochs: [500, 1000, 1500, 2000, 2500, 3000, 3500, 4000]
- Dropout: [0, 0.1, 0.2, 0.3, 0.4, 0.5]

- Learning rate: [0.001, 0.01, 0.1, 0.2, 0.3]
- Number of layers: [1, 2, 3, 4, 5]
- Hidden 1: [40, 64, 128, 256]
- Hidden 2: [40, 64, 128, 256]
- Activation: [Relu, Tanh, Softmax]
- Number of Mixtures: [1, 3, 5, 10, 15, 20]
- Optimizer: [Adam, SGD, RMSprop]
- Distribution of the target: [Normal, Log, Exponential]

The best hyper-parameter combination yielding the highest Accuracy and lowest error metrics is:

- Number of Epochs: [3000]
- Dropout: [0.4]
- Learning rate: [0.001]
- Number of layers: [2]
- Hidden 1: [128]
- Hidden 2: [128]
- Activation: [Relu]
- Number of Mixtures: [3]
- Optimizer: [Adam]
- Distribution of the target: [Normal]

15.2 Monte-Carlo Dropout

Monte-Carlo Dropout as mentioned in (Gal & Ghahramani, 2016) works by running multiple forward passes T through the model with a different dropout value (set the weight to zero) each time where it computes the predictive mean and variance out of the sampled network outputs. It applies a dropout mask during predictions/testing not only during the training phase and is one of the popular methods for modeling predictive uncertainty.

We have used the same framework that was developed by authors in (Gustafsson, Danelljan, & Schon, 2020) as they propose an evaluation framework to estimate epistemic uncertainty in deep learning. It is designed to test the robustness in real-world computer vision applications and is employed on the street-scene semantic segmentation (classification) and depth completion task (regression). Their proposed framework is also applied to toy classification and regression problems to provide the first properly extensive framework that compares the two current state-of-the-art scalable methods: MC-dropout and ensembling. Results have shown that ensembling consistently provides more reliable uncertainty estimates and always outperforms MC-dropout.

We have developed further evaluation of the uncertainty where we decomposed it into epistemic and aleatoric uncertainty and studied the impact of uncertainty while varying our input features and further developed more visual figures to demonstrate our comparison.

15.2.1 Architecture

The neural network is built using PyTorch. It consists of linear hidden layers having a ReLU activation and a dropout layer after each hidden layer. Then a final linear output layer is added which outputs the mean predictions and the model's uncertainty.

The scoring/loss function that we use is the negative log likelihood, which is defined as: where:

$$L(\theta, y) = -logP_\theta(y) \tag{15.2}$$

Equation 15.2: Taken from (Bloch, 2019)

- θ is the parameters of the distribution
- P_θ is the predicted probability distribution
- y is the real outcome

15.2.2 Hyper-parameters

We experimented with several experiments by varying some of the following hyper-parameters:

- Number of Epochs: The number of iterations the training data set is being shown to the neural network model while training.

- Batch Size: A mini-batch size is the number of samples we give to our model after the parameter update occurs.
- Dropout: A regularization technique that revolves around randomly dropping neurons so that their weights will not be updated resulting in improving generalizing and decreasing a model's ability to over-fit. It ranges from 0 to 0.5.
- Learning Rate: A tuning parameter that determines the step size at each iteration while moving toward a minimum of a loss function
- Number of layers: The number of hidden layers in our model between the input and output layers.
- Hidden 1: The number of hidden neurons for the first layer.
- Hidden 2: The number of hidden neurons for the second layer.
- Activation: A function that is attached to each neuron in the network and decides if a neuron should be fired or not based on its input if this input happens to be relevant to the model's prediction. It also aids in mapping the output to a range between 0 and 1 or -1 and 1.
- Optimizer: A function that is defined to reduce the loss by updating the weights or learning rates.

15.2.3 Implementation

We have built a neural network based on MC Dropout forward passes and employed hyper-parameters tuning and utility-based learning. The data set is divided into 70% training and 30% testing.

This module outputs testing scores, in addition to all necessary plots, figures, and data sets needed and found here

This module code is implemented using PyTorch. Our code is inspired by the Github code which is available here, but we have further tuned the scripts to match our implementation of providing a dynamic data set of two dimensions.

Our implementation is available here and divided into several files as such:

- datasets.py: This python file contains the processing of the training data set along with the processing of the testing data set.
- model.py: This file contains the neural network model architecture which further initializes each input, hidden, and output layer of a neural network.

- train.py: This file contains the training process of a neural network with outputting the performance of the network concerning the number of epochs. All hyper-parameters are defined here and we then save the network's weights and loss per epoch.
- eval.py: This file contains the evaluation process of the already trained network by loading the model's weights and providing the testing data set and then outputting the mean and different uncertainty types.

We tuned the MC Dropout hyper-parameters by utilizing a grid search and Utility-based regression. Each range of values for each hyper-parameter is chosen based on what is accepted in the literature.

- Number of Epochs: [100, 150, 200, 250, 300, 350, 400, 450, 500]
- Batch Size: [16, 32, 64, 128]
- Dropout: [0, 0.1, 0.2, 0.3, 0.4, 0.5]
- Learning rate: [0.001, 0.01, 0.1, 0.2, 0.3]
- Number of layers: [1, 2, 3, 4, 5]
- Hidden 1: [40, 64, 128, 256]
- Hidden 2: [40, 64, 128, 256]
- Activation: [Relu, Tanh, Softmax]
- Number of Mixtures: [1, 3, 5, 10, 15, 20]
- Optimizer: [Adam, SGD, RMSprop]

The best hyper-parameter combination yielding the highest Accuracy and lowest error metrics is:

- Number of Epochs: [150]
- Batch Size: [32]
- Dropout: [0.2]
- Learning rate: [0.001]
- Number of layers: [2]
- Hidden 1: [64]
- Hidden 2: [64]
- Activation: [Relu]
- Optimizer: [Adam]

15.3 Deep Ensemble

Regardless of the advantages of MC-Dropout, a few disadvantages likewise exist. For instance, the number of forward passes T of the network during forecast needs to stay sufficiently high, which can be computationally costly. Moreover, (Balaji Lakshminarayanan & Blundell, 2017) states that having a fixed dropout rate is also considered another limitation since it cannot be learned during the training process. This has motivated authors in (Balaji Lakshminarayanan & Blundell, 2017) to use Deep Ensemble as an alternative approach to model uncertainty.

Deep Ensemble is a group of neural network models used to predict uncertainty. It trains multiple individuals models with different parameter initialization. These models are treated as a uniformly-weighted mixture model, and they consolidate network outputs to yield the final forecasts like bagging(bootstrapping) as shown in **Figure 15.1**

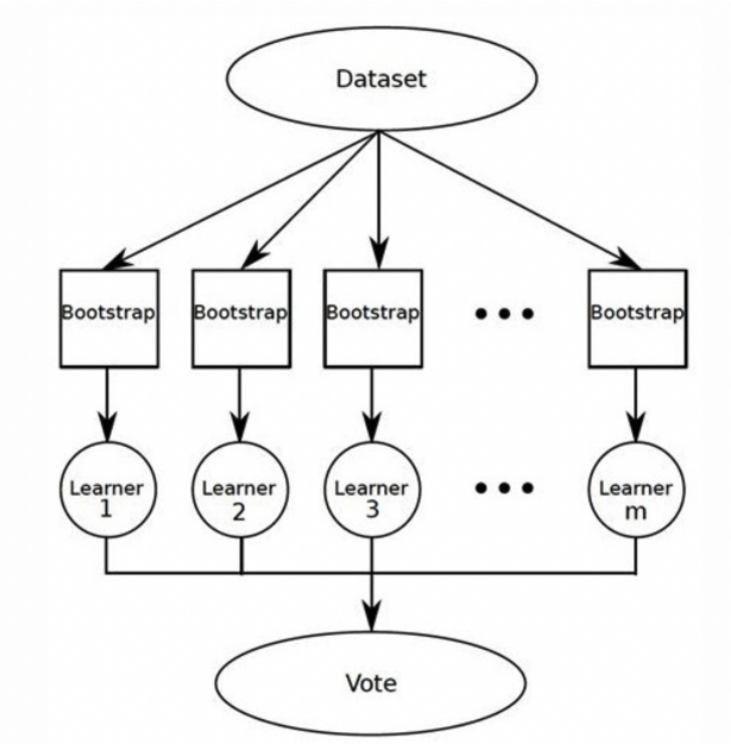


Figure 15.1: Deep Ensemble

We have used the author's implementation in (Gustafsson et al., 2020) and studied the impact of uncertainty on our model's prediction through different input combinations.

15.3.1 Architecture

The neural network is built using PyTorch. It consists of linear hidden layers having a ReLU activation and a dropout layer after each hidden layer. Then a final linear output layer is added which outputs the mean predictions and the model's uncertainty.

The scoring/loss function that we use is the negative log likelihood, which is defined as: where:

$$L(\theta, y) = -logP_\theta(y) \tag{15.3}$$

Equation 15.3: Taken from (Bloch, 2019)

- θ is the parameters of the distribution
- P_θ is the predicted probability distribution
- y is the real outcome

15.3.2 Hyper-parameters

We experimented with several experiments by varying some of the following hyper-parameters:

- Number of Epochs: The number of iterations the training data set is being shown to the neural network model while training.
- Learning Rate: A tuning parameter that determines the step size at each iteration while moving toward a minimum of a loss function
- Number of layers: The number of hidden layers in our model between the input and output layers.
- Hidden 1: The number of hidden neurons for the first layer.
- Hidden 2: The number of hidden neurons for the second layer.
- Activation: A function that is attached to each neuron in the network and decides if a neuron should be fired or not based on its input if this input happens to be relevant to the model's prediction. It also aids in mapping the output to a range between 0 and 1 or -1 and 1.
- Optimizer: A function that is defined to reduce the loss by updating the weights or learning rates.
- Number of the ensemble (M): The number of ensemble members.

15.3.3 Implementation

We have built a Deep Ensemble neural network consisting of five members and employed hyper-parameters tuning and utility-based learning. The data set is divided into 70% training and 30% testing.

This module outputs testing scores, in addition to all necessary plots, figures, and data sets needed and found here

This module code is implemented using PyTorch. Our code is inspired by the Github code which is available here, but we have further tuned the scripts to match our implementation of providing a dynamic data set of two dimensions.

Our implementation is available here and divided into several files as such:

- datasets.py: This python file contains the processing of the training data set along with the processing of the testing and validation data sets.
- model.py: This file contains the neural network model architecture which further initializes each input, hidden, and output layer of a neural network.
- train.py: This file contains the training process of a neural network with outputting the performance of the network concerning the number of epochs. All hyper-parameters are defined here and we then save the network's weights and loss per epoch.
- eval.py: This file contains the evaluation process of the already trained network by loading the model's weights and providing a holdout data set to ensure no over-fitting occurs.
- test.py: This file contains the testing process of the already trained network by loading the model's weights and providing the testing data set and then outputting the mean and different uncertainty types.

We tuned the Deep Ensemble hyper-parameters by utilizing a grid search and Utility-based regression. Each range of values for each hyper-parameter is chosen based on what is accepted in the literature.

- Number of Epochs: [100, 150, 200, 250, 300, 350, 400, 450, 500]
- Learning rate: [0.001, 0.01, 0.1, 0.2, 0.3]
- Number of layers: [1, 2, 3, 4, 5]
- Hidden 1: [40, 64, 128, 256]
- Hidden 2: [40, 64, 128, 256]

- Activation: [Relu, Tanh, Softmax]
- Number of Mixtures: [1, 3, 5, 10, 15, 20]
- Optimizer: [Adam, SGD, RMSprop]
- Number of ensemble: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

The best hyper-parameter combination yielding the highest Accuracy and lowest error metrics is:

- Number of Epochs: [150]
- Learning rate: [0.001]
- Number of layers: [2]
- Hidden 1: [64]
- Hidden 2: [64]
- Activation: [Relu]
- Optimizer: [Adam]
- Number of ensemble: 5

The best UBR and SmoGn parameters are:

- rel_method = “range”
- extr_type = “high”
- coef = 1.5
- relevance_pts = np.array([[1, 0 , 0],[4, 0 , 0], [15, 1 , 0]])
- thr_rel=0.1
- Cperc = np.array([1,1.2])
- repl = False
- dist = “Manhattan”
- p = 2
- pert = 0.1

More details about each UBR and SmoGn parameters are found in **Chapter** [**8**]

15.4 Results

We will present the testing results for all the clustering techniques we have performed and further analyze each. We have conducted Experiment A across Deep Ensemble, MC Dropout, EEflux (METRIC), and Stat on our daily data (union of clusters) and on different clusters (cluster by climate or cluster by seasons). We have conducted Experiment A on all the mentioned models across different feature selection scenarios (scenario A, scenario B, scenario C, and scenario D). However, we will first perform Experiment A between different probabilistic models to note the best performing model.

15.4.1 Choosing the Best Model

We have performed Experiment A on all our probabilistic models which we trained on all the input features ("Scenario A"). Data is split into 70% training and 30% testing and we further evaluated the results of each probabilistic model. This is performed to indicate the best performing model for us to proceed with our experimental variations using the best candidate. **Table [15.1]** shows the

Error Metrics	**MC Dropout Scores**	**Deep Ensemble Scores**	**MDN Scores**
Average	3.74715476	3.74715476	3.747154763
F1	0.95035504	0.94711805	0.9355302
F2	0.9286266	0.93252082	0.929859227
F05	0.97312467	0.96217955	0.941270768
Precision	0.98892041	0.97248953	0.945137113
Recall	0.91468468	0.92303676	0.926116621
R2	0.62293978	0.64621753	0.58741328
Adjusted R2	0.61449814	0.63829703	0.578176264
RMSE	1.1669836	1.130388	1.220722663
MSE	1.3618507	1.2777771	1.490163819
MAE	0.8501126	0.8284513	0.889568496
MAPE	30.1460988	28.0285631	29.9684217
Accuracy	69.8539012	71.9714369	70.0315783
Pearson C.C.	0.79176997	0.80621273	0.778911564
Spearman C.C.	0.8060766	0.818636	0.789701743
Spatial Distance	0.20823003	0.19378727	0.221088436
NMI	0.9990108	1	1
AIC	679.3	539.55	876.76
BIC	684.99	545.25	882.45
Data Size	10911	10911	10911

Table 15.1: Comparing probabilistic models

testing scores for the probabilistic models that we are studying. We note the following scores:

- MC Dropout scores an R2=0.6 and an Accuracy of 70%
- Deep Ensemble scores an R2=0.64 and an Accuracy of 72%
- MDN scores an R2=0.59 and an Accuracy of 70%

According to all the scores and error metrics, Deep Ensemble is the best performing model amongst other models.

As per **Table [15.1]**, we note that the results of MC Dropout and Deep Ensemble seem very comparable but yielding better results for the Deep Ensemble Model. Thus, we will perform experiments by sub-setting on all the climates and performing seasonality study i.e clustering by air temperature for the two models, and evaluate the results.

15.4.2 Utility-Based Learning and SmoGn Up-sampling

We have performed Experiment A on our best probabilistic model (Deep Ensemble) that is obtained from **Section [15.4.1]** and compared the result of this experiment to when we apply SmoGn up-sampling to our data set. SmoGn up-sampling is applied to our best model (MLP-Reptile). Details for the SmoGn hyper-parameters are found in **Section [15.3.3]**

	Metrics	Union of Clusters - Before SmoGn	Union of Clusters - After SmoGn
Models	**Average ET**	3.75	3.75
	Train Size	7638	7638
	Test Size	3273	3273
	Recall	0.93	0.82
	F2	0.92	0.81
	R2	0.67	0.52
	RMSE	1.15	1.32
MLP-Reptile	**MAE**	0.80	0.99
	Accuracy	74.51	64.11
	NMI	1.00	0.98
	Training Time (seconds)	511.57	2548.17
	Testing Time (seconds)	0.12	134.31
	Data size	10911	10911

Figure 15.2: Before and After SmoGn

Figure [13.5] portrays the experimental results for the union of clusters (all the data set) before SmoGn and after SmoGn using a Deep Ensemble model. Deep Ensemble with no SmoGn is performing better than Deep Ensemble with SmoGn by an R2 of 28.84%, Accuracy of 16.22%, Recall of 13.41%, and F2 of 13.58%. Up-sampling is adding more noise to our data set and thus the results

are not as good as before applying SmoGn. Thus we continue with using Deep Ensemble without any SmoGn techniques for any further experiments.

Takeaway Message SmoGn has left a negative impact on our best model Deep Ensemble, decreasing the Recall by 13.41%, the F2 measure by 13.58%, the R2 score by 28.84%, and the Accuracy by 16.22%.

15.4.3 Climate Study

We have conducted Experiment A across several models on our daily data (union of clusters) and on different climate subsets that are shuffled randomly with a split ratio of 70/30. The best feature selection scenario yielding the best Accuracy, Recall, and the least error metrics was "Scenario C". We will show the results for that scenario and all other scenarios will be presented in the appendix.

We have performed seven experiments where each experiment represents training and testing on a climate subset. Experiments are ordered from the least performing to the best performing. As per **Figure [15.3]**.
Experiments are defined as such:

- Cluster Dsa: We randomly train on climate Dsa using a 70/30 split
- Cluster Other: We randomly train on climate Other using a 70/30 split
- Cluster Cwa: We randomly train on climate Other using a 70/30 split
- Cluster Csb: We randomly train on climate Csb using a 70/30 split
- Cluster Csa: We randomly train on climate Csa using a 70/30 split
- Union of Clusters: We randomly train on all the climates using a 70/30 split
- Cluster Cfa: We randomly train on climate Cfa using a 70/30 split

	Clusters	Climate Dsa	Climate Other	Climate Cwa	Climate Csb	Climate Csa	Union of Climates	Climate Cfa
Models	Average ET	2.52	3.50	4.49	2.29	3.99	3.74	3.98
	Train Size	565	1167	472	431	2457	7638	2552
	Test Size	242	499	202	184	1052	3273	1093
	F2	0.53	0.00001	0.00001	0.00001	0.86	0.79	0.73
	Recall	0.51	0.00001	0.00001	0.00001	0.84	0.79	0.75
	R2	-0.12	-1.16	-1.62	-5.97	0.23	-0.78	-1.64
	RMSE	1.24	2.47	2.75	1.74	2.00	2.67	3.27
	MAE	1.10	1.93	2.03	1.45	1.40	2.01	2.57
	Accuracy	38.11	41.86	54.30	33.93	65.21	44.52	35.13
	NMI	0.86	0.89	0.90	0.96	0.95	0.87	0.83
	Training Time (seconds)	NA	NA	NA	NA	NA	NA	NA
EEflux (METRIC)	Testing Time (seconds)	NA	NA	NA	NA	NA	NA	NA
	F2	0.0001	0.90	0.97	0.00001	0.94	0.93	0.94
	Recall	0.90	0.88	0.97	0.00001	0.92	0.92	0.93
	R2	0.48	0.50	0.59	0.60	0.65	0.67	0.69
	RMSE	1.10	1.42	0.89	0.59	1.20	1.15	1.16
	MAE	0.80	1.02	0.61	0.45	0.82	0.80	0.82
	Accuracy	66.17	64.35	84.29	78.85	76.19	74.51	75.50
	NMI	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Training Time (seconds)	113.26	154.14	117.56	113.26	223.16	511.57	223.90
Deep Ensemble	Testing Time (seconds)	0.01	0.04	0.01	0.06	0.04	0.12	0.03
	F2	0.0001	0.90	0.96	0.00001	0.93	0.93	0.94
	Recall	0.89	0.87	0.96	0.00001	0.91	0.92	0.93
	R2	0.45	0.48	0.54	0.53	0.61	0.66	0.68
	RMSE	1.14	1.45	0.95	0.64	1.26	1.17	1.17
	MAE	0.78	1.05	0.66	0.49	0.87	0.83	0.83
	Accuracy	69.14	63.79	83.48	76.59	75.33	72.73	74.93
	NMI	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Training Time (seconds)	155.18	203.32	203.32	136.40	220.15	566.67	6751.83
	Testing Time (seconds)	0.01	0.15	0.01	0.05	0.32	0.77	0.05
MC Dropout	Data set Size	807	1666	674	615	3509	10911	3645

Figure 15.3: Sampling by Climate Results

As per **Figure** [**15.3**], the following is noted:

- Climate Dsa is the least performing climate
- Climate Csa shows comparable results to that of the union of clusters/climates
- Climate Cfa is the best-performing climate and shows better results than the union of climates with an R2=0.686 and Accuracy=75.5% for the Deep Ensemble Model and an R2=0.682 and Accuracy=74.9% for the MC Dropout Model
- Union of clusters shows an R2=0.669 and Accuracy=74.5% for the Deep Ensemble Model and an R2=0.660 and Accuracy=72.7% for the MC Dropout Model
- As been shown earlier in **Table** [**15.1**], Deep Ensemble is the best performing model for all climates not only for the union of climates but results for Climate Cfa are very comparable.

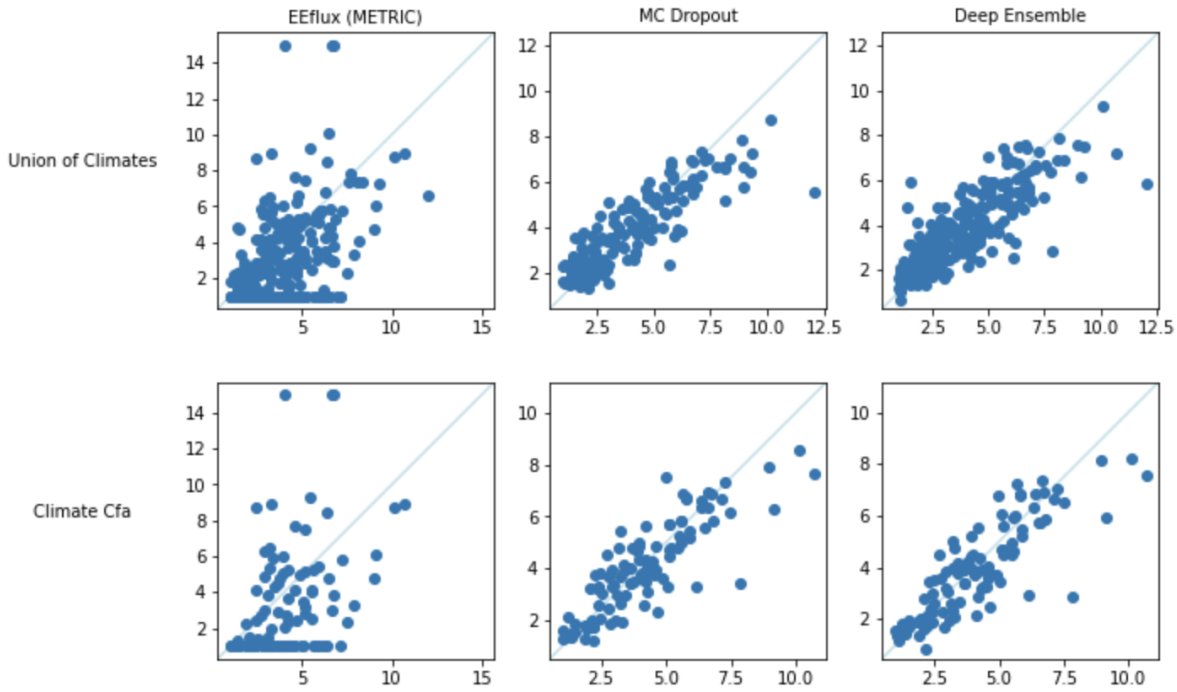


Figure 15.4: Experiment A Scatter Plot for Climate Study

Figure [15.4] illustrates on the x-axis the Real ET (mm) and on the y-axis the Predicted ET (mm). It shows the testing data set for MC Dropout (base model), Deep Ensemble (best model), and EEflux (Metric) model across the best climates and the union of climates ordered according to their performance. It is noticed that the Deep Ensemble shows a better diagonal fit than the MC Dropout model and EEFlux (METRIC) model across Climate Cfa and Union of Climates. The points in the EEflux (METRIC) model were scattered and not centered around the bisector. Deep Ensemble trained on Climate Cfa yielded a better concentration around the bisector in comparison to the union of climates.

We have further conducted a residual analysis study for the union of clusters and the best performing climates (climate Cfa) by studying the performance of Experiment B, Experiment C, and Experiment D across different models as shown in **Figure [15.5]**

Figure [15.5] shows the experimental results across the best climate cluster and its union. We contrast the performance of the Deep Ensemble and the MC Dropout Models in minimizing the residual biases as mentioned in Experiments B, C, and D. We note the following:

	Clusters	Union of Clusters			Climate Cfa		
Models	Average ET	3.74623853			3.98847666		
	Residuals	Experiment B	Experiment C	Experiment D	Experiment B	Experiment C	Experiment D
	F2	0.90	0.95	0.90	0.91	0.95	0.91
	Recall	0.91	0.96	0.91	0.92	0.96	0.92
	R2	0.83	0.78	0.83	0.88	0.87	0.88
	RMSE	0.26	1.14	0.26	0.25	1.12	0.25
	MAE	0.16	0.82	0.16	0.15	0.81	0.15
	Accuracy	76.25	NA	NA	75.88	NA	NA
	NMI	1	1	1	1	1	1
	Training Time (seconds)	NA	NA	NA	NA	NA	NA
Deep Ensemble	Testing Time (seconds)	NA	NA	NA	NA	NA	NA
	F2	0.88	0.95	0.88	0.92	0.95	0.92
	Recall	0.88	0.96	0.88	0.92	0.96	0.92
	R2	0.90	0.83	0.90	0.89	0.88	0.89
	RMSE	0.30	0.75	0.85	0.77	0.74	0.77
	MAE	0.17	1.12	0.30	0.24	1.08	0.24
	Accuracy	77.15	0.82	0.17	0.14	0.79	0.14
	NMI	1.00	1.00	1.00	78.11	NA	NA
	Training Time (seconds)	1	1	1	1	1	1
MCDropout	Testing Time (seconds)	NA	NA	NA	NA	NA	NA
	F2	-	-	-	-	-	-
	Recall	-	-	-	-	-	-
	R2	-	-	-	-	-	-
	RMSE	-	-	-	-	-	-
	MAE	-	-	-	-	-	-
	Accuracy	-	-	-	-	-	-
	NMI	-	-	-	-	-	-
	Training Time (seconds)	-	-	-	-	-	-
	Testing Time (seconds)	-	-	-	-	-	-
Stat	Data set Size	10,911	10,911	10,911	3,645	3,645	3,645

Figure 15.5: Residual Analysis for Climate Study

- MC Dropout outperforms all others in minimizing the absolute bias across the union of clusters, producing an R2 score of 0.83, and MAE of 1.12
- MC Dropout outperforms all others in minimizing the proportional bias across the union of clusters, producing an R2 score of 0.90, MAE of 0.30, and accuracy of 77.15%
- MC Dropout outperforms all others in minimizing the combined bias across the union of clusters, producing an R2 score of 0.9, and MAE of 0.30
- Experiment B was the best in terms of all the reported metrics compared to Experiment C and Experiment D. Thus, minimizing the proportional bias was the most successful residual analysis

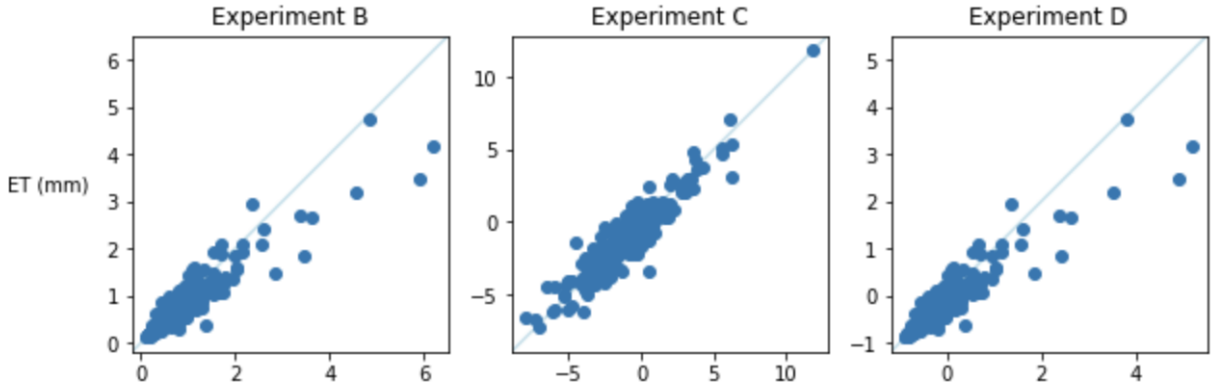


Figure 15.6: Residual Analysis Scatter Plots for Climate Study

In **Figure [15.6]**, the columns represents Experiments B, C, and D respectively and the row represents the two variables as been defined for each experiment in **Chapter [12]** for Deep Ensemble Model as being our best model for the union of climates. We note that Experiments B and D show very comparable visualizations having the points scattered around the bisector with a bit more distribution as opposed to Experiment C. Experiment C show a better diagonal than the others. However, quantitative results as shown in **Figure [15.5]** indicate that MLP-Reptile was the best point-wise model in minimizing the proportional bias, which is represented by Experiment B.
N.B: AutoML results are not reported as earlier since we could not extract the testing data set for us to compute the above metrics.

Takeaway Message: Deep Ensemble was the best probabilistic model to predict Real ET. Deep Ensemble beats MC Dropout in R2 by 0.58% to 1.36% and

in RMSE by 0.68% to 0.94%. Deep Ensemble performs best on climate Cfa (Humid Subtropical - mild with a dry season and hot summer), which is considered suitable for farmers would need to know irrigation factor values the most in this type of climate, rather than a climate with rich irrigation and moist season. Deep Ensemble trained on climate Cfa beats the union of climates in R2 by 2.5%, in Accuracy by 1.34%, and in Recall by 0.86%. **Figure [15.4]** confirms our quantitative observations present in **Figure [15.3]**.

15.4.4 Seasonality Study

We have conducted Experiment A across Deep Ensemble, MC Dropout, EEflux (METRIC), and Stat on our daily data (union of clusters) and on different seasons where each experiment represents training and testing on the same season. The best feature selection scenario yielding the best Accuracy, Recall, and the least error metrics was "Scenario C". We will show the results for that scenario and all other scenarios will be presented in the appendix.
We have studied the model's performance in each separate season. There exist three clusters where each belongs to a season.
Cluster 0 represents summer, Cluster 1 represents winter, and Cluster 2 represents spring.

We have performed three experiments. Experiments were defined as such:

- Winter: We randomly train and test on the winter season data set using a 70/30 split. This data set is obtained by clustering on TA using Kmeans, $k = 3$ on the third cluster
- Spring: We randomly train and test on the spring season data set using a 70/30 split. This data set is obtained by clustering on TA using Kmeans, $k = 3$ on the second cluster
- Summer: We randomly train and test on the summer season data set using a 70/30 split. This data set is obtained by clustering on TA using Kmeans, $k = 3$ on the first cluster
- Union of Clusters: We randomly train on all the seasons using a 70/30 split

	Clusters	Winter		Spring		Summer		Union of Clusters	
Models	**Average ET**	2.32873773		3.59964093		4.95491741		3.74623853	
	Train Size	2173		3510		3853		7638	
	Test Size	930		1,504		1,651		3,273	
	Data sets	Validation	Testing	Validation	Testing	Validation	Testing	Validation	Testing
	F2	NA	-	NA	-	NA	0.00001	NA	0.79
	Recall	NA	-	NA	-	NA	0.00001	NA	0.79
	R2	NA	-	NA	-	NA	-2.81	NA	-0.78
	RMSE	NA	-	NA	-	NA	3.73	NA	2.67
	MAE	NA	-	NA	-	NA	2.82	NA	2.01
	Accuracy	NA	-	NA	-	NA	44.35	NA	44.52
	NMI	NA	-	NA	-	NA	0.91	NA	0.87
	Training Time (seconds)	NA	NA	NA	NA	NA	NA	NA	NA
EEflux (METRIC)	**Testing Time (seconds)**	NA	NA	NA	NA	NA	NA	NA	NA
	F2	0.00001	0.00001	0.00001	0.92	0.95258539 +- 0.005	0.94	0.91971708 +- 0.006	0.93
	Recall	0.00001	0.87	0.00001	0.90	0.94989474 +- 0.006	0.94	0.89825806 +- 0.010	0.92
	R2	0.29446581 +- 0.05	0.34	0.40	0.49	0.65071812 +- 0.03	0.56	0.68636934 +- 0.0155	0.67
	RMSE	1.2425307 +- 0.04	1.01	1.00	1.20	1.1765621 +- 0.04	1.42	1.2579134 +- 0.030	1.15
	MAE	0.8735857 +- 0.03	0.73	0.71	0.89	0.8879817 +- 0.037	1.06	0.82197434 +- 0.017	0.80
	Accuracy	70.3756755 +- 1.07	67.45	68.92	69.80	65.9157068 +- 1.88	69.72	72.5975055 +- 1.08	74.51
	NMI	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Training Time (seconds)	500.57	500.57	576.02	576.02	218.21	218.21	511.57	511.57
Deep Ensemble	**Testing Time (seconds)**	0.23	0.23	0.04	0.47	0.29	0.29	0.12	0.12
	F2	0.00	0.00	0.00	0.00	0.95262973 +- 0.004	0.94	0.92634674 +- 0.0009	0.93
	Recall	0.00	0.85	0.00	0.86	0.94541887 +- 0.004	0.94	0.91003093 +- 0.0012	0.92
	R2	0.25108739 +- 0.017	0.32	0.30785995 +- 0.03	0.34	0.65077549 +- 0.03	0.51	0.62330324 +- 0.006	0.66
	RMSE	1.2801583 +- 0.014	1.03	1.2306799 +- 0.03	1.01	1.1764655 +- 0.06	1.49	1.2413518 +- 0.009	1.17
	MAE	0.88716304 +- 0.013	0.72	0.86948097 +- 0.03	0.72	0.9167441 +- 0.044	1.12	0.86196303 +- 0.006	0.83
	Accuracy	70.6880409 +- 0.54	69.52	69.5291483 +- 1.26	68.19	64.4176989 +- 2.26	66.70	72.2507273 +- 0.186	72.73
	NMI	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Training Time (seconds)	299.44	299.44	271.15	271.15	361.78	361.78	566.67	566.67
MCDropout	**Testing Time (seconds)**	0.25	0.25	0.29	0.29	0.40	0.40	0.77	0.77
	F2	-	-	-	-	-	-	-	-
	Recall	-	-	-	-	-	-	-	-
	R2	-	-	-	-	-	-	-	-
	RMSE	-	-	-	-	-	-	-	-
	MAE	-	-	-	-	-	-	-	-
	Accuracy	-	-	-	-	-	-	-	-
	NMI	-	-	-	-	-	-	-	-
	Training Time (seconds)	-	-	-	-	-	-	-	-
	Testing Time (seconds)	-	-	-	-	-	-	-	-
Stat	**Data set Size**	3,103	3,103	5,014	5,014	5,504	5,504	10,911	10,911

Figure 15.7: Seasonality

As per **Figure [15.7]**, the following is noted:

- The winter season (Cluster 1) was the worst-performing as opposed to the other clusters/seasons with an R2=0.34 and an Accuracy=67.4% for the Deep Ensemble Model and an R2=0.31 and an Accuracy=69.5% for the MC Dropout Model.

- The summer season (Cluster 0) was the best performing as opposed to the other seasons with an R2=0.55 and an Accuracy=69.71% for the Deep Ensemble Model and an R2=0.51 and an Accuracy=66.7% for the MC Dropout Model.

We have further conducted a residual analysis study for the union of clusters and the best performing season (Summer) by studying the performance of Experiment B, Experiment C, and Experiment D across different models as shown in **Figure [15.8]**

Figure [15.8] shows the experimental results across the best seasonality cluster (summer) and its union. We contrast the performance of the Deep Ensemble and the MC Dropout Models in minimizing the residual biases as mentioned in Experiments B, C, and D. We note the following:

	Clusters	Summer			Union of Clusters		
Models	Average ET	4.95491741			3.74623853		
	Residuals	Experiment B	Experiment C	Experiment D	Experiment B	Experiment C	Experiment D
	F2	0.85	0.95	0.85	0.90	0.95	0.90
	Recall	0.85	0.95	0.85	0.91	0.96	0.91
	R2	0.84	0.86	0.84	0.83	0.78	0.83
	RMSE	0.28	1.30	0.28	0.26	1.14	0.26
	MAE	0.18	1.01	0.18	0.16	0.82	0.16
	Accuracy	79.21	NA	NA	76.25	NA	NA
	NMI	1.00	1.00	1.00	1	1	1
	Training Time (seconds)	NA	NA	NA	NA	NA	NA
Deep Ensemble	Testing Time (seconds)	NA	NA	NA	NA	NA	NA
	F2	0.82	0.94	0.82	0.88	0.95	0.88
	Recall	0.82	0.95	0.82	0.88	0.96	0.88
	R2	0.81	0.83	0.81	0.90	0.83	0.90
	RMSE	0.31	1.43	0.31	0.85	0.75	0.85
	MAE	0.19	1.14	0.19	0.30	1.12	0.30
	Accuracy	77.50	NA	NA	0.17	0.82	0.17
	NMI	1.00	1.00	1.00	77.15	NA	NA
	Training Time (seconds)	NA	NA	NA	1	1	1
MCDropout	Testing Time (seconds)	NA	NA	NA	NA	NA	NA
	F2	-	-	-	-	-	-
	Recall	-	-	-	-	-	-
	R2	-	-	-	-	-	-
	RMSE	-	-	-	-	-	-
	MAE	-	-	-	-	-	-
	Accuracy	-	-	-	-	-	-
	NMI	-	-	-	-	-	-
	Training Time (seconds)	-	-	-	-	-	-
	Testing Time (seconds)	-	-	-	-	-	-
Stat	Data set Size	5,504	5,504	5,504	10,911	10,911	10,911

Figure 15.8: Residual Analysis for Seasonality Study

- Deep Ensemble outperforms all others in minimizing the absolute bias across the summer season, producing an R2 score of 0.86, and MAE of 1.01
- Deep Ensemble outperforms all others in minimizing the proportional bias across the summer season, producing an R2 score of 0.84, MAE of 0.18, and accuracy of 79.21%
- Deep Ensemble outperforms all others in minimizing the combined bias across the union of clusters, producing an R2 score of 0.84, and MAE of 0.18
- Experiment B was the best in terms of all the reported metrics compared to Experiment C and Experiment D. Thus, minimizing the proportional bias was the most successful residual analysis

Takeaway Message: Deep Ensemble was the best probabilistic model to predict Real ET. Deep Ensemble beats MC Dropout in R2 by 0.58% to 1.36% and in RMSE by 0.68% to 0.94%. The best performing season was the summer season with an R2=0.55 and an Accuracy=69.71%.

15.5 Uncertainty Quantification

After examining with several probabilistic models and noting that Deep Ensemble was our best model in terms of yielding the highest Accuracy and lowest error metrics. We will further confirm that by comparing MC Dropout and Deep Ensemble using uncertainty metrics for "Scenario C" as we have used in the previous sections.

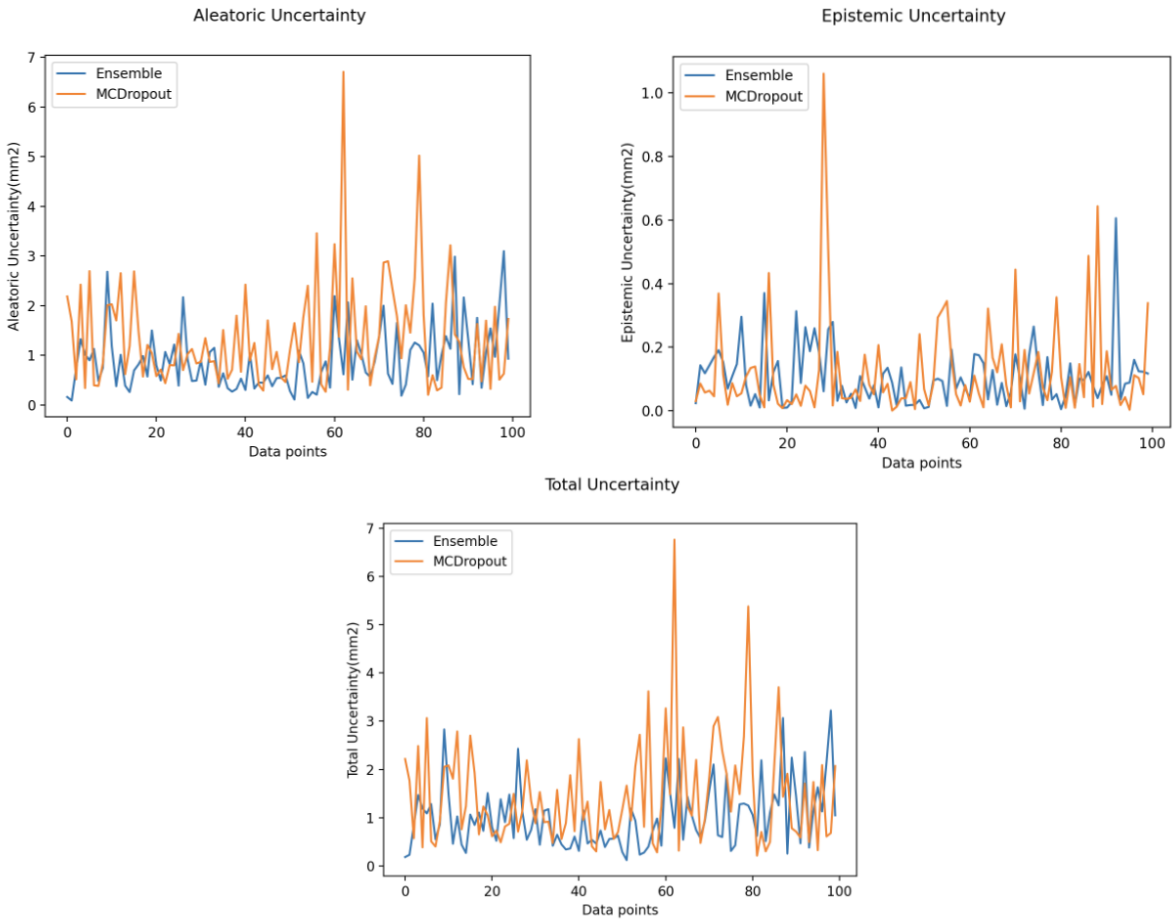


Figure 15.9: Uncertainty Quantification between MC Dropout and Deep Ensemble

Figure [15.9], shows on the x-axis 100 testing data points for the MC-Dropout (in orange), and Deep Ensemble models (in blue). The y-axis shows the aleatoric, epistemic, or total uncertainty in (mm^2). As shown in **Figure [15.9]**, the aleatoric, epistemic, and total uncertainty is higher for the MC Dropout Model meaning MC Dropout Model is less certain than Deep Ensemble.

Thus we will continue with studying with different feature selection methods utilizing our best probabilistic and most certain model i.e Deep Ensemble.

We have experimented with four different experimental variations, each pertaining to a feature selection method, utilizing Deep Ensemble as mentioned in **Table [15.2]**.

Combinations	Model	Name	Input Combinations
1	DeepEnsemble1	Scenario A	All Columns
2	DeepEnsemble2	Scenario B	TA(5 lags)
3	DeepEnsemble3	Scenario C	TA(5 lags) WS(2 lags) RH(3 lags) EEflux LST(5 lags) EEflux NDVI(2 lags) EEflux Albedo(2 lags)
4	DeepEnsemble4	Scenario D	TA(5 lags) RH(3 lags) EEflux LST(5 lags)

Table 15.2: Feature Selection Scenarios

Models	Metrics	Standard Deviation Validation	Validation Scores	Testing Scores
Deep Ensemble1	Recall	0.00798687	0.89880748	0.92287661
	F2	0.00514443	0.91605511	0.93288075
	R2	0.01579645	0.67335243	0.65560319
	RMSE	0.03028143	1.2837521	1.1772883
	Accuracy	1.00480863	70.9085482	74.1304804
	Training Time (seconds)	NA	NA	505.57238
	Testing Time (seconds)	NA	NA	0.07739592
Deep Ensemble2	Recall	0.00578996	0.88924413	0.91049342
	F2	0.00510831	0.91291401	0.92412707
	R2	0.01095162	0.6243729	0.58659515
	RMSE	0.01980728	1.3766387	1.2898555
	Accuracy	0.93372673	69.5943482	71.1985056
	Training Time (seconds)	NA	NA	470.6844
	Testing Time (seconds)	NA	NA	0.06228161
Deep Ensemble3	Recall	0.01048178	0.89825806	0.92338169
	F2	0.006626	0.91971708	0.93430161
	R2	0.01557407	0.68636934	0.6698199
	RMSE	0.0302498	1.2579134	1.152733
	Accuracy	1.0857688	72.5975055	74.5091767
	Training Time (seconds)	NA	NA	511.56939
	Testing Time (seconds)	NA	NA	0.12408829
	Recall	0.00762346	0.84030658	0.8765349
	F2	0.00570801	0.86924625	0.89724382
	R2	0.01117238	0.25798637	0.33982586
	RMSE	0.01447867	1.9348506	1.6299812
	Accuracy	1.50555374	50.6338111	56.7280494
	Training Time (seconds)	NA	NA	505.5
	Testing Time (seconds)	NA	NA	0.11339784
	Training Dataset Size	NA	NA	7638
Deep Ensemble4	Testing Dataset Size	NA	NA	3273

Table 15.3: Feature Selection Experiments

Table [15.3].

When comparing different error and accuracy measures, it is noted that Deep-Ensemble3 was the best performing amongst all as shown in **Table [15.3]** having R2 = 0.67, Accuracy = 74.5% and Recall = 0.92.
Our best feature selection method is the one named "DeepEnsemble3".

We will further want to quantity the epistemic, aleatoric, and total uncertainty for all the feature selection methods and analyze which one is the least uncertain or the most certain.

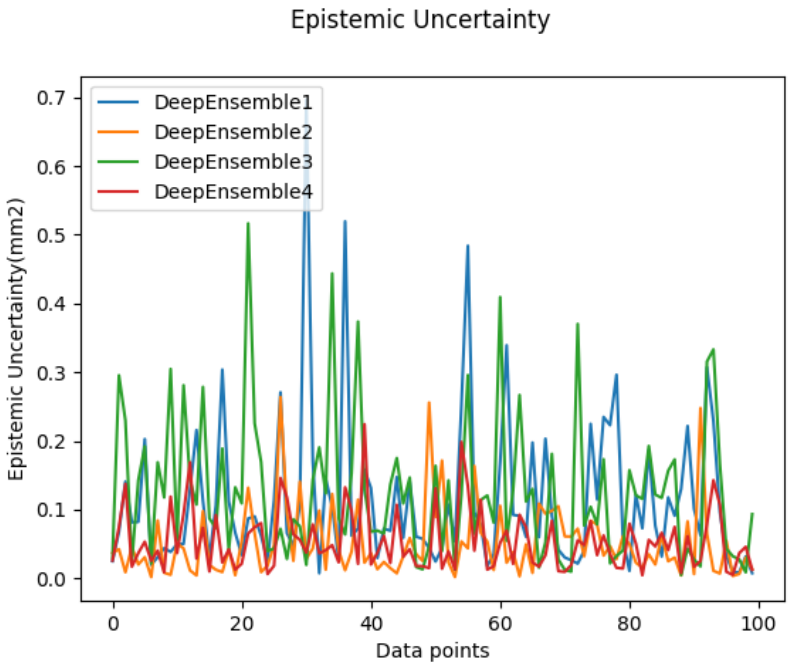


Figure 15.10: Epistemic Uncertainty

Figure [15.10] represents the epistemic uncertainty for 100 data points of the testing data set. The x-axis represents the number of data points and the y-axis represents the epistemic uncertainty in mm^2 which is of the same unit as our output variable.

As shown in **Figure [15.10]**, we note the following:

- DeepEnsemble1 was the most uncertain
- DeepEnsemble4 was the least uncertain
- DeepEnsemble3 seems to fall between DeepEnsemble1 and DeepEnsemble4
- DeepEnsemble2 shows analogous results to that of DeepEnsemble4

The reason for having DeepEnsemble2 and DeepEnsemble4 to be the least uncertain since they have fewer input combinations meaning they are less complex in terms of their dimensions. However, DeepEnsemble1 and DeepEnsemble3 are less certain since they have higher input combinations i.e higher dimension.

In addition to that, it is noted that as the data points increase from 10 to 100 data points, the red line (DeepEnsemble4) for instance decreases which validates the theory behind the epistemic uncertainty in the sense that with more data, the epistemic uncertainty decreases.

Moving to studying the aleatoric uncertainty as shown in **Figure [15.11]**, we note the following:

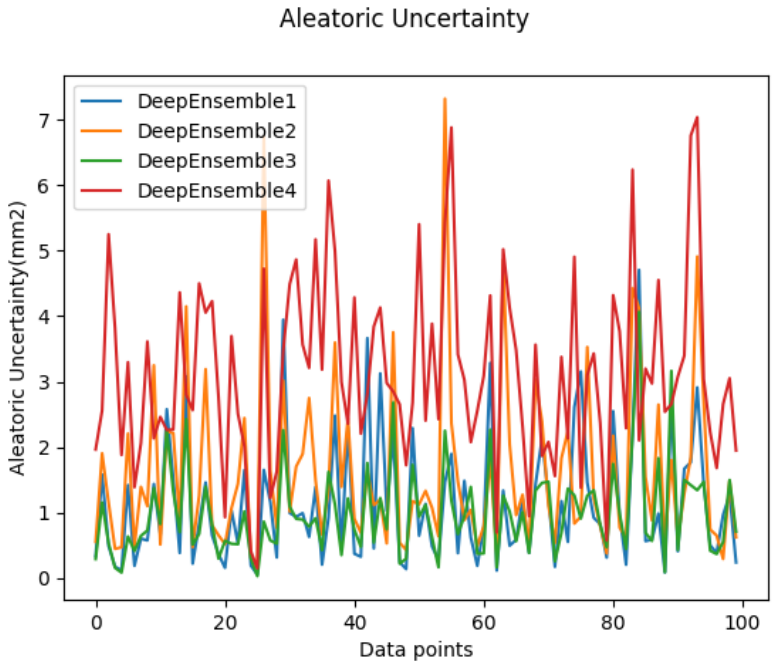


Figure 15.11: Aleatoric Uncertainty

- DeepEnsemble4 was the most uncertain
- DeepEnsemble1 and DeepEnsemble3 were the least uncertain
- DeepEnsemble2 shows analagous results to that of DeepEnsemble4

We note a different behavior of what we visualized earlier for the epistemic uncertainty. As we increase the problem dimension (i.e moving from DeepEnsemble4 to DeepEnsemble2 to DeepEsemble3 to DeepEnsemble1), as we are moving from a lower feature selection dimension to a higher dimension, the epistemic uncertainty increases and the aleatoric uncertainty decreases as shown and explained in **Figure [15.11]** and **Figure [15.10]**. Moreover, the reason epistemic uncertainty is increasing is because fitting a model will become more difficult and require more data when the problem dimension increases.

We will further study the impact of the total uncertainty as shown in **Figure [15.12]**

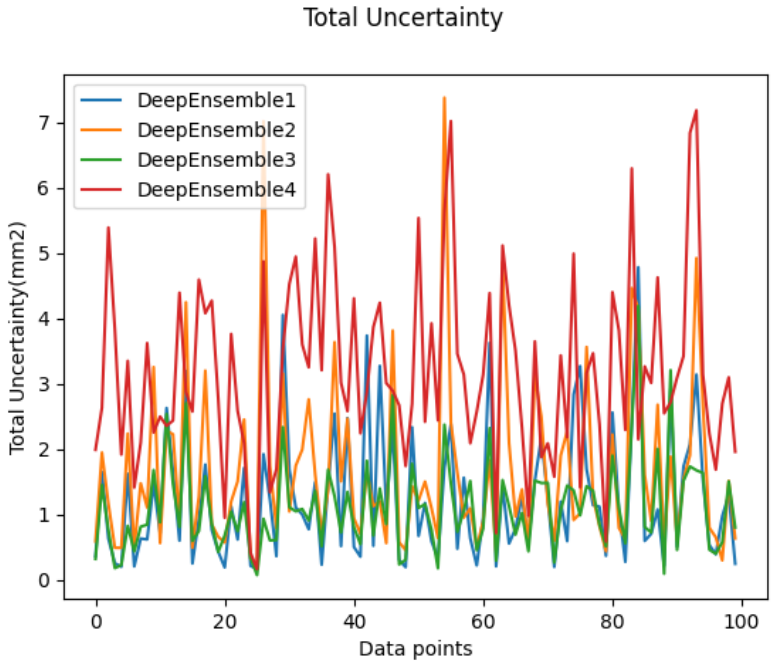


Figure 15.12: Total Uncertainty

We note the following:

- The most uncertain model is DeepEnsemble4 followed by DeepEnsemble2.
- DeepEnsemble1 and DeepEnsemble3 were the most certain showing analogous results.

15.6 Takeaway Message

After performing climate study and seasonality study using Deep Ensemble and MC Dropout, we note that our best performing model was Deep Ensemble with an R2 = 0.669 and Accuracy = 74.5% for the union of clusters and with an R2 = 0.68 and an Accuracy = 75.5% for climate Cfa which was the best-performing climate. Furthermore, we did not benefit much from the cluster by air temperature (seasonality study) as opposed to the climate subset study has been indicated earlier.

After computing different error and accuracy metrics for several feature selection methods and studying the impact of the uncertainty on each, the best model we chose is "DeepEnsemble3" as has been shown and explained in **Figure [15.12]** as it is the most certain model.

Moreover, it is also noted that our uncertainty quantification confirms what we have as being the best model in **Table** [**15.3**].

In **Figure** [**15.12**], DeepEnsemble1 shows comparable results to DeepEnsemble3, but since it has higher dimensions i.e more input features as per **Table** [**15.2**], its epistemic uncertainty is higher as shown in **Figure** [**15.10**].

15.7 Models' Stability and Performance

For all our experiments, we have evaluated our testing data set on 100 repetitions in order to yield consistent and stable results rather than random ones. We will compare the performance of our models in terms of different metrics i.e (the most accurate, the most certain, the least training time, and the one yielding the least bias and variance).

15.7.1 Most Accurate Model

When comparing our models in terms of accuracy, it is noted that Deep Ensemble is the best probabilistic model with an accuracy of 74.51% as opposed to the MC Dropout model yielding an accuracy of 72.73%.

15.7.2 Most Precise Model

When comparing our models in terms of utility-based metrics, it is noted that Deep Ensemble is the best probabilistic model with a recall of 0.923, and precision of 0.978 as opposed to the MC Dropout model yielding a recall of 0.916, and precision of 0.99. Thus the Deep Ensemble model is a better candidate in capturing correctly rare from non-rare values as it has a higher recall than an MC Dropout model.

15.7.3 Most Certain Model

When comparing our models in terms of uncertainty, it is noted that MC Dropout shows high fluctuations as shown in figure **Figure** [**15.9**] for all the uncertainty types. However, Deep Ensemble shows low uncertainty values and more stable values for each uncertainty type. Thus, the most certain model is the Deep Ensemble model.

15.7.4 Least Training Time Model

When comparing the training time of our models, MC Dropout took 566.67 seconds for learning with 200 epochs, however, Deep Ensemble took 511.57 seconds with 200 epochs.

15.7.5 Learning Experience

We have computed validation scores, validation standard deviation, and evaluated our model on a shuffled testing data set for each fold.

- For MC Dropout Model: We have computed validation scores and plotted the learning curve.
  - It yields an accuracy of 72.73% for the testing data set, an accuracy of 72.2% for the validation data set, and a validation standard deviation of 0.186.
  - It yields an MSE of 1.37 mm for the testing data set, an MSE of 1.54 mm for the validation data set, and a validation standard deviation of 0.02.

  Thus the training and validation scores are very comparable for MC Dropout; meaning the validation data set is tracking well the training data set.

- For Deep Ensemble Model: We have computed validation scores and plotted the learning curve.
  - It yields an accuracy of 74.51% for the testing data set, an accuracy of 72.5% for the validation data set, and a validation standard deviation of 1.08.
  - It yields an MSE of 1.32 mm for the testing data set, an MSE of 1.58 mm for the validation data set, and a validation standard deviation of 0.07.

  Thus the training and validation scores are very comparable for Deep Ensemble; meaning the validation data set is tracking well the training data set.

Chapter 16

SHAP

16.1 Background

In numerous utilizations of AI, users are usually inquired to trust a model for it would help them to make better choices, but a "doctor will not simply operate on a patient simply because the model said so." (Marco Tulio Ribeiro & Guestrin, 2016). This is also applicable for lower-stakes events such as picking a movie to watch from Netflix; we'll need to have some kind of trust in our model. Trust could be built by understanding the rationale behind a model's prediction. For instance, as been shown in **Figure [16.1]**, that the model predicts that a certain patient has the flu and the prediction is further explained by an "explainer" i.e (SHAP or LIME) that sheds lights on the symptoms that are most relevant to the model. Having this information about the model, the doctor is simply capable of deciding whether the model can be trusted or not. SHAP (Shapley

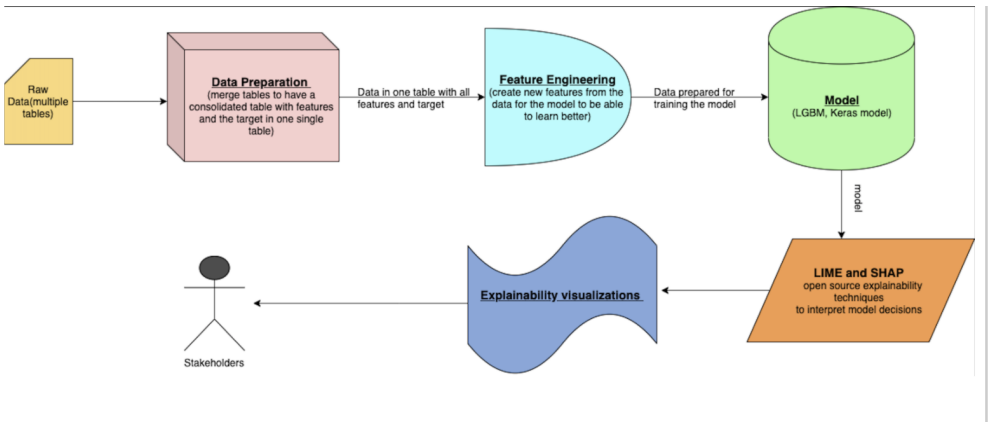


Figure 16.1: Explaining a model's prediction

Additive exPlanations) is a technique for interpreting a model's prediction and being able to explain its output. Each feature has a SHAP value which indicates its importance or how much it contributes to a particular prediction.

SHAP value is the "average of the marginal contributions across all permutations." SHAP values as been explained in (Parsa & Movahedi, 2020) has several benefits:

- Global Interpretability: A set of SHAP values can show how each predictor often contributes i.e negatively or positively to the output feature. In the case of classification, SHAP shows if a low or high value of an input feature contributes to increasing or decreasing the probability of the output variable belonging to a certain class. In the case of regression, SHAP shows if a low or high value of an input feature contributes to increasing or decreasing the value of an output variable. The SHAP interpretation is examined on the whole data set.
- Local Interpretability: Each observation has a collective SHAP value. The SHAP interpretation is examined on a subset of the data. Local interpretability helps us define and contrast the effects of the input features on the target variable. This is important because SHAP allows us to study each instance of any input variable, and analyze its effects on our model prediction.
- Flexibility: It can be computed for any tree or neural network-based model. Keras and Tensorflow models are supported by SHAP by using a Deep Explainer. DeepExplainer is a class specialized for computing SHAP values for neural network models.

16.1.1 Dealing with Categorical Data

Our data set consists of several encoded columns i.e Site Id, Month, and Vegetation. Each column is encoded and hence is split into several columns having a value of 0 or 1 which is why it is difficult to indicate the importance of the Site Id or the other columns as a whole. To represent the encoded columns as one column we have performed the same strategy as mentioned by the author of the SHAP package Lundberg and thus did the following:

- Fit our neural network model
- Compute the SHAPely value array for each data point in our data frame.
- Sum up the SHAPely values corresponding to the encoded parts of each column. Example (Sum up the SHAPely values for Site Id 1, Site Id 2, Site Id 3) resulting in 1 value only which will indicate the importance of that column as a whole
- Perform an OR operation on the encoded column parts to produce one column for each encoded column (Parsa & Movahedi, 2020)

This will result in having 1 column which has the SUM of the SHAPely values instead of having all its parts.

16.2 Point-wise and Probabilistic SHAP

SHAP supports a wide range of plots and each has its own purpose, we will restrict our study to summary bar plots, summary plots, and decision plots.

We perform SHAP analysis on our best point-wise and probabilistic models which are based on a multi-layer perceptron neural network. The same analysis is reflected for both since SHAP takes a neural network estimator.

16.2.1 SHAP Summary Bar Plot

In **Figure** [**16.2**], the x-axis represents the mean SHAPely value for each variable and the y-axis represents the input features. A SHAPely value will determine the impact of each input feature on our predictions in the sense of how much does the input contributes to decreasing or increasing the value of the output feature. It does not show us if this shift is a positive or a negative, or if this shift caused the Predicted ET to be closer to the Real ET. As shown in **Figure** [**16.2**], TA is the largest contributor which shifted the output variable by an average of 0.8 mm.

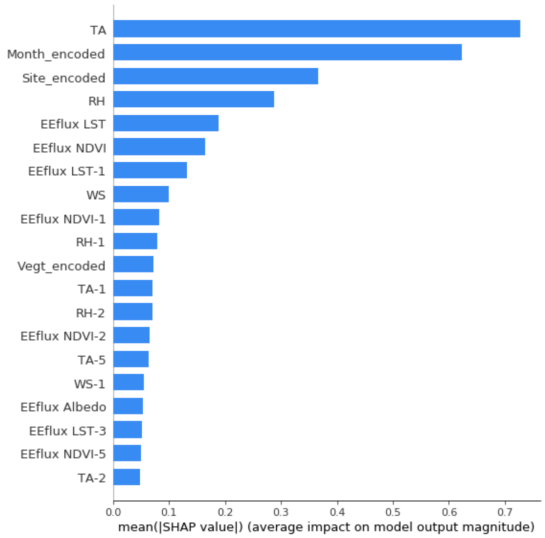


Figure 16.2: Summary Bar Plot

16.2.2 SHAP Summary Plot

This plot is often more descriptive than **Figure [16.2]** since it shows the negative and positive relationships of the predictors with the target variable. This plot shows the following information:

- Feature Importance: Input features are ranked in a descending order with respect to their importance of contributing to the output feature.
- Impact: The horizontal location shows whether the effect of that value is associated with a higher or lower prediction.
- Original Value: A red color indicates whether a variable's value is high and blue color indicates whether a variable's value is low. A mix of red and blue shows that the input variable value is around the average.

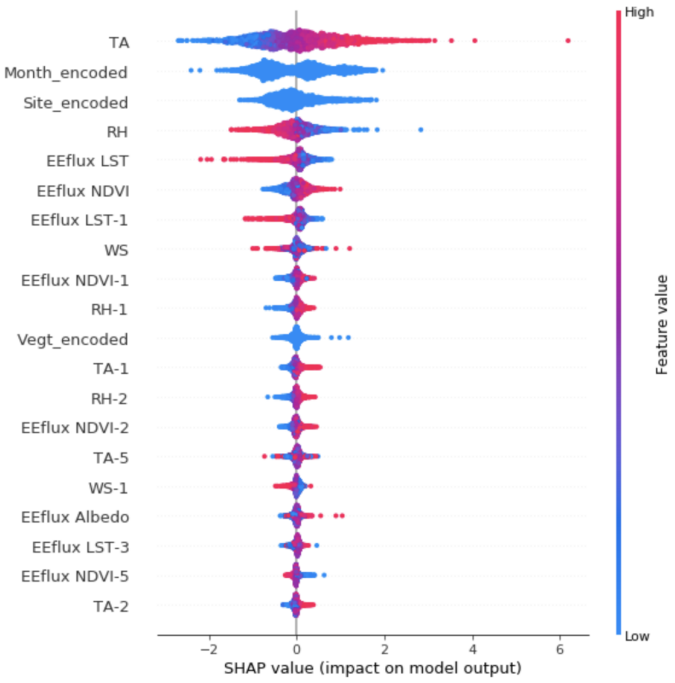


Figure 16.3: Summary Plot

From **Figure [16.3]**, we analyze that TA is the highest contributing feature for our model. When TA has high values, the SHAPely values are high (hence the model output value increased) followed by EEflux LST and TA having the same analysis results. LST values, for example, yield higher SHAPely values when they are low and lower SHAPely values when they are high. Month and site, since they are encoded variables, cannot be measured by their value. It can only be mentioned that they rank second and third in terms of contribution respectively.

16.2.3 SHAP Decision Plot for All Predictions

In **Figure [16.4]**, the x-axis represents the model's output and the y-axis lists the input features ordered by their importance in a descending order. The top and bottom vertical bands represent the model output value (real ET). Each prediction is represented by a colored line indicating its importance. At the top of the plot, each line strikes the x-axis at its corresponding observation's predicted value. This value determines the color of the line on a spectrum. Moving from the bottom of the plot to the top, SHAPely values for each feature are added to the model's base value (a value that would be predicted if the model wasn't exposed to the top contributing features). This shows how each feature contributes to the overall prediction. **Figure [16.4]** represents the decision plot on a global

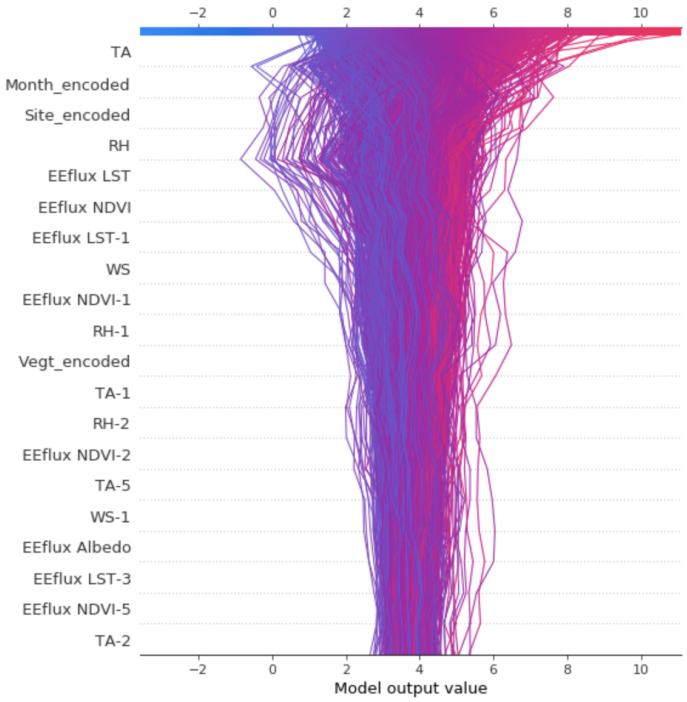


Figure 16.4: Summary Decision Plot Global

level i.e for all the predictions.
As it is observed, the model output value before introducing the major contributing input variables was restricted between 2 and 5 mm, then this range witnessed a big shift upon introducing EEflux NDVI, EEflux LST, RH, Site, Month, and TA. The red color denotes a high-value influence (introduction of a variable yielded in a higher value Real ET), and the blue color denotes a low-value influence (introduction of a variable yielded in a lower value Real ET). For instance, TA has a negative impact on ET values from 0 to 4, causing a decrease in the predicted ET value. However, TA has a positive impact on ET values that are greater than

4, causing a decrease in the predicted ET value.

16.3 Observations

- High values of TA imply high values of predicted ET.
- High values of EEflux NDVI imply high values of predicted ET.
- Low values of RH imply high values of predicted ET.
- Low values of WS imply high values of predicted ET.
- Low values of EEflux LST imply high values of predicted ET.
- Low or high values of EEflux Albedo does not indicate a certain range or pattern for ET.
- TA, EEflux LST, and RH are the top contributing features

All of our observations appear to be in concurrence with what irrigation specialists consider to be valid. We would also like to shed light on the fact that the observations we noted did not change when we tested SHAP on the most accurate or rare predictions.

Chapter 17

LIME

17.1 Background

LIME, a local interpretable model-agnostic explanation, is a technique used to explain the predictions of any machine learning model in an interpretable manner and evaluate its usefulness in various tasks related to trust (Marco Tulio Ribeiro & Guestrin, 2016). LIME is a surrogate model that explains an individual prediction of a black-box machine learning model by tweaking the input slightly and further testing the changes in prediction. The tweak should be small so that we are still close to our original data point. In other words, LIME models the impact of the changes in the prediction based on the changes to the input. We often use surrogate models if a model is too complex to test. Hence, the surrogate model is a basic model that imitates complex model mechanisms. Surrogate models are usually a linear regression or decision tree trained on a complex model's original inputs and predictions.

LIME approximates a black-box model by a simple model locally i.e (one in the neighborhood of the prediction we want to explain), rather than approximating a model globally. A local interpretation is considered essential since it shows how each data point is affected individually by the input features. The authors of LIME argued that locality is a special aspect of LIME, for it is important to analyze each test point individually rather than make a generalization.

The steps taken to train a local surrogate model are as follows:

- Select a testing point that we wish to interpret its result
- LIME creates a data set of permutations out of the selected point and gets the black-box predictions for these points
- LIME weights the newly generated data set concerning their proximity (using Euclidean distance) to the selected point

- Interpret the prediction by analyzing the local model

17.2 Observations

We applied LIME after we split our data by 70/30 ratio and performed shuffling. LIME uses a base linear regression model. We further define a LIME tabular explainer, which takes in the following as input:

- The training data
- A random testing data point
- The numerical columns: Wind Speed, Relative Humidity, Air Temperature, EEflux LST, EEflux Albedo, and EEflux NDVI
- The categorical columns: Site, Month, and Vegetation
- The number of top contributing features to portray: 10

17.2.1 LIME on an Inaccurate Data Point

We interpret the results for an inaccurate randomly selected test point (RMSE $\geq$ 2) at a local level in **Figure [17.1]**. It is noted that the Predicted ET value for this specific input data point is 0.6 mm. The following observations are noted:

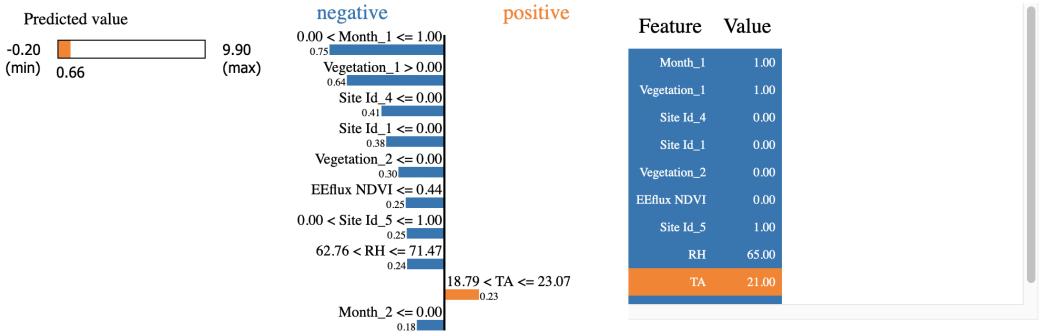


Figure 17.1: LIME plot for an inaccurate point

- TA values between 18.79 C and 23.07 C are highly positively correlated with the output variable Real ET.
- EEflux NDVI values less than 0.44 are highly negatively correlated with the output variable Real ET.
- RH values between 62.76 and 71.47 are highly negatively correlated with the output variable Real ET.

17.2.2 LIME on an Two Accurate Data Points

We aim to portray how LIME's locality is useful by analyzing the interpretation across two accurate test predictions.

We interpret the results for the first accurate randomly selected test point (RMSE ≤ 1) at a local level in **Figure [17.2]**. It is noted that the Predicted ET value for this specific input data point is 1.05 mm. The following observations are noted:

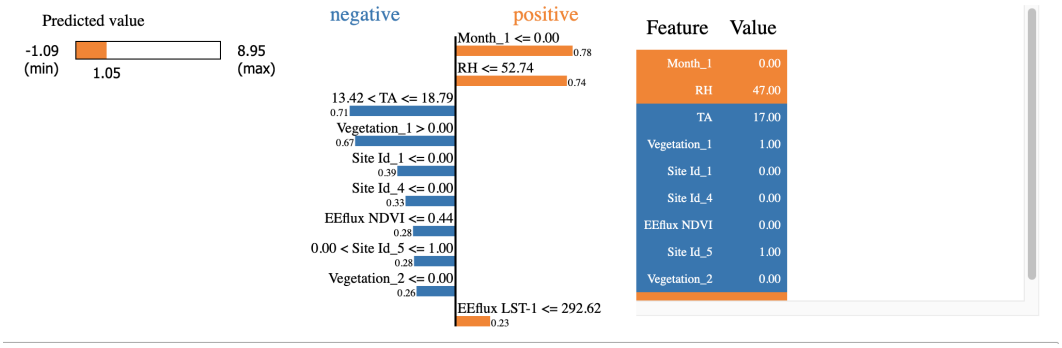


Figure 17.2: LIME plot for an accurate point

- TA values between 13.42 C and 18.79 C are highly negatively correlated with the output variable Real ET.
- EEflux NDVI values less than 0.44 are highly negatively correlated with the output variable Real ET.
- RH values less than 52.74 are highly positively correlated with the output variable Real ET.

We further analyze the results for another accurate selected test point at a local level as well in **Figure [17.3]**. It is noted that the Predicted ET value for this specific input data point is 1.69 mm. The following observations are noted:

- TA values $\leq$ 13.43 C are highly negatively correlated with the output variable Real ET.
- EEflux LST values $\leq$ 292 are highly negatively correlated with the output variable Real ET.
- RH values $\geq$ 71.35 are highly negatively correlated with the output variable Real ET.

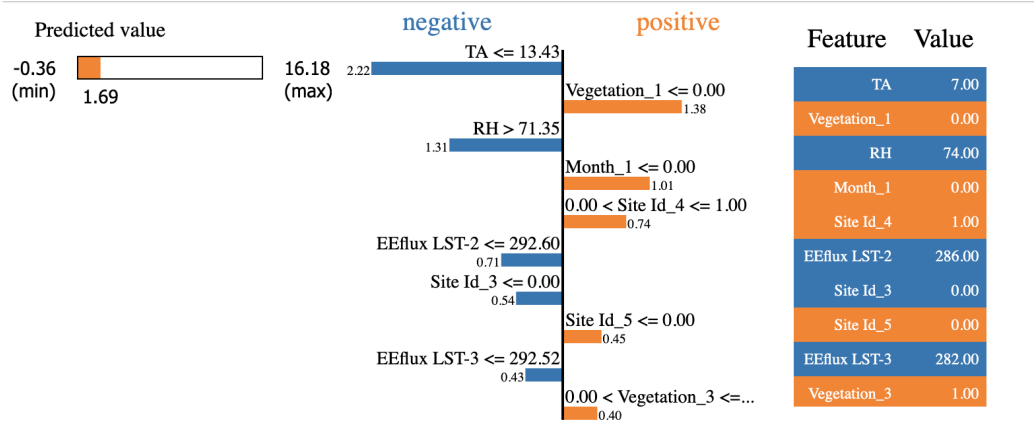


Figure 17.3: LIME plot for another accurate point

It is noticed that even when the chosen local points are accurate points and fall within the same range of ET, they do not yield the same interpretation for all our features. They do much in the analysis for TA for instance when TA where values $\leq$ 13.43 C are highly negatively correlated with ET. This highlights the importance of locality in LIME where we can zoom in into each test point instead of generalizing across all the data points.

17.2.3 Comparison between LIME on Accurate and Inaccurate Data Points

We note that for the first two observations, the numerical results of course differ but the trend if somehow similar:

- For inaccurate predictions, TA values between 13.42 C and 18.79 C have a negative correlation with the output Real ET, however, for accurate predictions, TA values between 18.79 C and 23.07 C have a positive correlation with the output Real ET
- For accurate and inaccurate predictions, NDVI values greater than 0.44 have a positive correlation with the output Real ET.

However, both **Figure [17.1]** and **Figure [17.2]** show only local interpretations, the values may change if we select different data points. Thus, we cannot generalize based on these two points only.

17.3 Comparison between LIME and SHAP

- Both are model agnostic and are surrogate models in the sense that they use the black-box machine learning models and tweak the input slightly by

creating permutations of the data set to test the effect of this change on the prediction. For instance, if we have a sentence as an input, we create a new sentence by removing or adding a word. This becomes a new permuted sentence.

- LIME is faster than SHAP since it perturbs the data around a single prediction to build a model whereas, SHAP has to compute all the permutations globally.
- SHAP does not have an optimized module to support all types of algorithms; however, LIME allows so.
- SHAP offers global and local interpretability, unlike LIME which allows only local interpretability.

17.4 Comparison between SHAP, LIME and Feature Selection

It is vital to differentiate between feature selection methods and inter- portability methods. Feature selection is a method that uses statistical techniques to determine how a subset of input features contributes to the target variable, thereby improving the precision of the predicted variable values. Interpretability techniques, however, describe the values that led to changing the value of the target variable (be it a good change towards higher accuracy or a bad change towards lower accuracy). Interpretability tools allow the user to uncover the mystery behind the predictions of a black-box model, whereas the feature selection methods simply highlight the set of input features that show more accurate results. Nevertheless, both approaches shed light on the most critical input features to be used in the training of a machine learning model. In our experiments, both interpretability methods (SHAP and LIME) and feature selection show that the top contributing features are TA, LST, RH, and NDVI.

Chapter 18

Deployment

After treating our problem as a regression problem and experimenting with different machine learning models, we further aim into building trust with our models using different interpretability tools i.e (SHAP and LIME). Finally, we'll have to deploy our models somewhere (cloud solution) to perform real-time predictions and feed our machine learning with more data with time.

There are several areas that one needs to consider that are related to the decision-making process of the deployment procedure.

- Storing and retrieving data: Data could be stored either on-site, in distributed storage (cloud solution), or a blend of the two. It bodes well to store the data where the model training takes place and the outcomes/predictions will be delivered. Henceforth, on-site model training and maintenance will be more suited for on-site data, especially if the information is critical, while cloud-based data stored in distributed storage frameworks, for example, GCS, AWS S3, or Azure should be accompanied by cloud model training.

- Frameworks and tools: Our models won't train, operate, and dispatch on their own. We need the right frameworks and tools combined with the relevant software and hardware that would assist us with deploying our ML models safely. For training our models, the frameworks we utilized are, for example, PyTorch, tensor-flow, Keras, and scikit-learn. These frameworks offer the three significant aspects in deployment: popularity, efficiency, and support.

- Feedback and iteration: It is essential to get feedback from a model that is under development. In case of model output deterioration, bias creep, or even data skew, actively following and controlling model status will notify us regarding any issue. This would ensure that before the end-client notices, such concerns are effortlessly settled. A new model to be deployed should

be checked appropriately and on a regular interval basis. In addition to that, continuous integration should be employed for it continuously tests and deploy new models without interrupting the current model processes.

There are many more factors that often affect the deployment process. We however chose to focus on the three most essential ones, and our choice of a framework varies depending on our business problem since often there is a trade-off between these three.

Chapter 19

Conclusion and Future Work

We have built a robust meta-learning reptile module that predicted ET (mm) successfully with an R2 of 0.79 - 0.88, an Accuracy of 76% - 90%, and F2 of 0.96 - 0.98 across different climates and union of climates. MLP-Reptile has proven to best the best point-wise model for it beat the MLP model as well as EEflux (METRIC) model. Moreover, we also experimented with different probabilistic and uncertainty models such as MC Dropout and Deep Ensemble. Deep Ensemble was the best probabilistic model that achieved an R2 of 0.66 - 0.69, an Accuracy of 74.5% to 75.5%, and an F2 of 0.93 - 0.94 across different climates and union of climates. We resorted to interpretability techniques using SHAP and LIME for us to explore the inner workings of our models. We also applied different feature selection techniques to identify the most important input features in obtaining the fastest training and the least economically expensive model, along with the best performing model. Both have shown analogous results for having TA, LST, and RH to be the top contributing. That is being said, as per our study we have offered two solutions to our users. One yielding a point prediction estimation for Real ET (mm) achieving an Accuracy of 76% and minimizing the proportional bias between Real ET and EEflux (METRIC) ET with an Accuracy of 81.32%. However, sometimes a point prediction is often restrictive and we want the user to have access to a valid range or probabilistic interval, thus the other solution yields a probabilistic prediction for Real ET (mm) achieving an Accuracy of 75.5% and minimizing the proportional bias between Real ET and EEflux (METRIC) ET with an Accuracy of 79.21%.

Future work would be to evolve and tune our models more and also incorporate uncertainty into our meta-learning reptile module for it being our best-performing model. This will help in measuring our model in terms of how much we are certain not just in terms of accuracy and other error metrics. We will also want to incorporate our best-performing models into a robust and real-time mobile application that would be user-friendly for agricultural specialists, or farmers to use especially for seasons requiring more irrigation i.e (summer, or spring).

Appendix A

Point-wise Experiments

This chapter includes all of the experimental results that were done but not mentioned in the original report. We have performed our experiments on all of our base (MLP) and best (MLP-Reptile) point-wise models across all the feature selection Scenarios (A,B,C, and D) mentioned in **Table [15.2]**. This chapter includes the following:

1. Experiments performed using MLP model on all the data set and the clusters in **Section [A.1]**

2. Experiments performed using MLP-Reptile model on all the data set and the clusters in **Section [A.2]**

A.1 MLP Point-wise Results

A.1.1 Scenario A

Error Metrics	Csb Testing	Csb Validation	Dsa Testing	Dsa Validation	Cwa Testing	Cwa Validation
Mean Target	2.28	-	2.41	-	4.53	-
F1	0.00002	-	0.00002	-	0.00002	-
F2	0.00005	-	0.00005	-	0.00005	-
F05	0.0000125	-	0.0000125	-	0.0000125	-
Precision	0.00001	-	0.00001	-	0.00001	-
Recall	0.85	-	0.88	-	0.95	-
R2	0.35	-	0.42	-	0.60	-
Adjusted R2	0.11	-	0.27	-	0.47	-
RMSE	0.73	-	1.08	-	0.84	-
MSE	0.54	0.48	1.16	1.31	0.70	0.83
MAE	0.52	0.52	0.77	0.88	0.67	0.66
MAPE	26.21	23.93	35.07	40.37	17.15	16.74
Accuracy	73.79	76.07	64.93	59.63	82.85	83.26
Pearson C.C.	0.60	-	0.65	-	0.80	-
Spearman C.C.	0.67	-	0.60	-	0.80	-
Spatial Distance	0.40	-	0.35	-	0.20	-
NMI	1.00	-	1.00	-	1.00	-
AIC	23183.07	-	23333.60	-	23226.60	-
BIC	60633.86	-	63976.23	-	61764.61	-
Data Size	184	-	242	-	202	-
Training (s)	376.60	376.60	178.19	178.19	686.48	686.48
Testing (s)	0.88	0.88	1.72	1.72	0.77	0.77

Table A.1: Scenario A of Sampling By Climate - Part 1

Error Metrics	Union of Clusters Testing	Union of Clusters Validation	Cfa Testing	Cfa Validation
Mean Target	3.73	-	4.00	-
F1	0.95	-	0.96	-
F2	0.94	-	0.94	-
F05	0.96	-	0.98	-
Precision	0.97	-	1.00	-
Recall	0.94	-	0.93	-
R2	0.67	-	0.69	-
Adjusted R2	0.66	-	0.67	-
RMSE	1.12	-	1.16	-
MSE	1.25	1.40	1.34	1.41
MAE	0.82	0.83	0.82	0.81
MAPE	26.64	26.38	25.05	24.67
Accuracy	73.36	73.62	74.95	75.33
Pearson C.C.	0.82	-	0.83	-
Spearman C.C.	0.82	-	0.85	-
Spatial Distance	0.18	-	0.17	-
NMI	1.00	-	1.00	-
AIC	24028.60	-	23619.99	-
BIC	95011.35	-	81826.33	-
Data Size	3273	-	1093	-
Training (s)	280.67	280.67	209.59	209.59
Testing (s)	0.22	0.22	0.14	0.14

Table A.2: Scenario A of Sampling By Climate - Part 2

A.1.2 Scenario B

Error Metrics	Dsa Testing	Dsa Validation	Csb Testing	Csb Validation	Cwa Testing	Cwa Validation
Mean Target	2.55	-	2.24	-	4.65	-
F1	0.00002	-	0.00002	-	0.00002	-
F2	0.00005	-	0.00005	-	0.00005	-
F05	0.0000125	-	0.0000125	-	0.0000125	-
Precision	0.00001	-	0.00001	-	0.00001	-
Recall	0.89	-	0.85	-	0.94	-
R2	0.23	-	0.31	-	0.53	-
Adjusted R2	0.16	-	0.23	-	0.48	-
RMSE	1.30	-	0.77	-	0.96	-
MSE	1.70	2.09	0.59	0.57	0.92	1.14
MAE	0.99	1.07	0.58	0.57	0.75	0.83
MAPE	43.31	45.45	30.06	27.19	18.23	24.75
Accuracy	56.69	54.55	69.94	72.81	81.77	75.25
Pearson C.C.	0.49	-	0.57	-	0.77	-
Spearman C.C.	0.48	-	0.68	-	0.73	-
Spatial Distance	0.51	-	0.43	-	0.23	-
NMI	1.00	-	1.00	-	1.00	-
AIC	19586.74	-	19362.17	-	19440.36	-
BIC	53530.62	-	50640.28	-	51626.50	-
Data Size	242	-	184	-	202	-
Training (s)	132.77	132.77	342.66	342.66	418.28	418.28
Testing (s)	0.66	0.66	0.72	0.72	1.23	1.23

Table A.3: Scenario B of Sampling By Climate - Part 1

Error Metrics	Csa Testing	Csa Validation	Union of Clusters Testing	Union of Clusters Validation	Cfa Testing	Cfa Validation
Mean Target	4.02	-	3.75	-	4.05	-
F1	0.95	-	0.95	-	0.95	-
F2	0.94	-	0.93	-	0.93	-
F05	0.96	-	0.98	-	0.97	-
Precision	0.96	-	0.99	-	0.98	-
Recall	0.93	-	0.91	-	0.92	-
R2	0.54	-	0.60	-	0.64	-
Adjusted R2	0.54	-	0.59	-	0.63	-
RMSE	1.31	-	1.27	-	1.32	-
MSE	1.71	1.89	1.61	1.71	1.75	1.61
MAE	0.96	0.96	0.92	0.93	0.93	0.90
MAPE	29.48	29.59	30.40	31.09	28.83	29.40
Accuracy	70.52	70.41	69.60	68.91	71.17	70.60
Pearson C.C.	0.74	-	0.77	-	0.80	-
Spearman C.C.	0.75	-	0.78	-	0.83	-
Spatial Distance	0.26	-	0.23	-	0.20	-
NMI	1.00	-	1.00	-	1.00	-
AIC	20019.91	-	21009.27	-	20068.14	-
BIC	68269.90	-	80292.56	-	68680.86	-
Data Size	1053	-	3273	-	1093	-
Training (s)	195.09	195.09	291.65	291.65	205.51	205.51
Testing (s)	0.71	0.71	2.51	2.51	0.70	0.70

Table A.4: Scenario B of Sampling By Climate - Part 2

A.1.3 Scenario C

Error Metrics	Dsa Testing	Dsa Validation	Csb Testing	Csb Validation	Cwa Testing	Cwa Validation	Csa Testing	Csa Validation
Mean Target	2.49	-	2.27	-	4.35	-	4.05	-
F1	0.00002	-	0.00001	-	0.00002	-	0.94	-
F2	0.00005	-	0.00001	-	0.00005	-	0.93	-
F05	0.0000125	-	0.00001	-	0.0000125	-	0.96	-
Precision	0.00001	-	0.00001	-	0.00001	-	0.96	-
Recall	0.90	-	0.00	-	0.95	-	0.93	-
R2	0.41	-	0.51	-	0.56	-	0.60	-
Adjusted R2	0.30	-	0.38	-	0.45	-	0.59	-
RMSE	1.15	-	0.66	-	0.98	-	1.22	-
MSE	1.31	1.50	0.43	0.55	0.96	0.75	1.50	1.55
MAE	0.84	0.88	0.52	0.54	0.76	0.67	0.88	0.89
MAPE	36.51	36.03	25.59	25.89	23.31	19.33	25.65	26.93
Accuracy	63.49	63.97	74.41	74.11	76.69	80.67	74.35	73.07
Pearson C.C.	0.67	-	0.74	-	0.78	-	0.78	-
Spearman C.C.	0.57	-	0.76	-	0.80	-	0.79	-
Spatial Distance	0.33	-	0.26	-	0.22	-	0.22	-
NMI	1.00	-	1.00	-	1.00	-	1.00	-
AIC	21955.87	-	21734.72	-	21882.76	-	22315.10	-
BIC	60142.29	-	56922.19	-	58091.75	-	76595.72	-
Data Size	242	-	184	-	202	-	1053	-
Training (s)	168.30	168.30	285.46	285.46	425.94	425.94	156.61	156.61
Testing (s)	0.96	0.96	1.28	1.28	0.69	0.69	0.56	0.56

Table A.5: Scenario C of Sampling By Climate - Part 1

Error Metrics	Union of Clusters Testing	Union of Clusters Validation	Cfa Testing	Cfa Validation	Other Testing	Other Validation
Mean Target	3.80	-	4.00	-	3.51	-
F1	0.96	-	0.96	-	0.95	-
F2	0.94	-	0.94	-	0.91	-
F05	0.98	-	0.97	-	0.99	-
Precision	0.99	-	0.99	-	1.02	-
Recall	0.93	-	0.93	-	0.89	-
R2	0.65	-	0.68	-	0.51	-
Adjusted R2	0.65	-	0.67	-	0.47	-
RMSE	1.17	-	1.20	-	1.41	-
MSE	1.38	1.37	1.43	1.25	1.98	2.02
MAE	0.82	0.82	0.82	0.81	1.01	1.03
MAPE	25.67	26.66	25.42	24.09	36.47	35.79
Accuracy	74.33	73.34	74.58	75.91	63.53	64.21
Pearson C.C.	0.81	-	0.83	-	0.72	-
Spearman C.C.	0.82	-	0.85	-	0.70	-
Spatial Distance	0.19	-	0.17	-	0.28	-
NMI	1.00	-	1.00	-	1.00	-
AIC	22939.00	-	22281.02	-	22231.00	-
BIC	89631.95	-	76969.70	-	68337.98	-
Data Size	3273.00	-	1093.00	-	499.00	-
Training (s)	260.79	260.79	244.82	244.82	131.95	131.95
Testing (s)	1.18	1.18	1.19	1.19	0.16	0.16

Table A.6: Scenario C of Sampling By Climate - Part 2

A.1.4 Scenario D

Error Metrics	Csb Testing	Csb Validation	Dsa Testing	Dsa Validation	Csa Testing	Csa Validation
Mean Target	2.24	-	2.52	-	4.05	-
F1	0.00001	-	0.00002	-	0.92	-
F2	0.00001	-	0.00005	-	0.89	-
F05	0.00001	-	0.00001	-	0.96	-
Precision	0.00001	-	0.00001	-	0.98	-
Recall	0.00001	-	0.88	-	0.87	-
R2	0.19	-	0.35	-	0.34	-
Adjusted R2	0.11	-	0.30	-	0.33	-
RMSE	0.83	-	1.21	-	1.65	-
MSE	0.69	0.76	1.48	1.40	2.71	2.32
MAE	0.66	0.71	0.96	0.88	1.24	1.20
MAPE	35.33	36.60	45.62	39.52	38.33	37.45
Accuracy	64.67	63.40	54.38	60.48	61.67	62.55
Pearson C.C.	0.45	-	0.60	-	0.58	-
Spearman C.C.	0.49	-	0.54	-	0.57	-
Spatial Distance	0.55	-	0.40	-	0.42	-
NMI	0.99	-	1.00	-	1.00	-
AIC	19132.96	-	19296.18	-	20252.48	-
BIC	49999.56	-	52793.47	-	67867.67	-
Data Size	184.00	-	242.00	-	1053.00	-
Training (s)	179.19	179.19	133.80	133.80	180.75	180.75
Testing (s)	0.49	0.49	0.52	0.52	0.33	0.33

Table A.7: Scenario D of Sampling By Climate - Part 1

Error Metrics	Union of Clusters Testing	Union of Clusters Validation	Cwa Testing	Cwa Validation	Cfa Testing	Cfa Validation
Mean Target	3.71	-	4.47	-	4.00	-
F1	0.93	-	0.00002	-	0.95	-
F2	0.90	-	0.00005	-	0.92	-
F05	0.96	-	0.00001	-	0.98	-
Precision	0.98	-	0.00001	-	1.00	-
Recall	0.88	-	0.94	-	0.90	-
R2	0.37	-	0.50	-	0.51	-
Adjusted R2	0.37	-	0.45	-	0.50	-
RMSE	1.56	-	1.08	-	1.48	-
MSE	2.43	2.43	1.16	1.07	2.18	2.57
MAE	1.18	1.17	0.82	0.80	1.10	1.18
MAPE	41.67	40.61	22.25	19.78	35.62	38.35
Accuracy	58.33	59.39	77.75	80.22	64.38	61.65
Pearson C.C.	0.61	-	0.75	-	0.72	-
Spearman C.C.	0.63	-	0.80	-	0.74	-
Spatial Distance	0.39	-	0.25	-	0.28	-
NMI	1.00	-	1.00	-	1.00	-
AIC	22112.93	-	19231.47	-	20055.95	-
BIC	80616.26	-	50994.15	-	68029.09	-
Data Size	3273.00	-	202.00	-	1093.00	-
Training (s)	336.06	336.06	630.42	630.42	173.30	173.30
Testing (s)	1.11	1.11	0.33	0.33	0.40	0.40

Table A.8: Scenario D of Sampling By Climate - Part 2

A.2 MLP-Reptile Point-wise Results

A.2.1 Scenario A

Error Metrics	Climate Csb	Climate Dsa	Climate Other	Union of Climates	Climate Csa	Climate Cwa	Climate Cfa
Mean Target	2.31	2.58	3.51	3.75	3.98	4.50	3.97
F1	0.00002	0.00002	0.94	0.95	0.96	0.98	0.96
F2	0.00005	0.00005	0.96	0.96	0.96	0.98	0.97
F05	0.0000125	0.0000125	0.92	0.94	0.97	0.98	0.96
Precision	0.00001	0.00001	0.91	0.93	0.97	0.98	0.96
Recall	0.86	0.90	0.97	0.97	0.96	0.98	0.97
R2	0.43	0.52	0.68	0.70	0.79	0.85	0.85
Adjusted R2	0.23	0.39	0.64	0.70	0.78	0.80	0.84
RMSE	0.74	0.98	1.08	1.08	0.91	0.59	0.82
MSE	0.55	0.96	1.16	1.16	0.83	0.35	0.67
MAE	0.54	0.71	0.85	0.81	0.69	0.44	0.60
MAPE	30.63	32.15	33.25	29.50	20.54	11.58	18.29
Accuracy	69.37	67.85	66.75	70.50	79.46	88.42	81.71
Pearson C.C.	0.75	0.73	0.88	0.87	0.89	0.92	0.92
Spearman C.C.	0.76	0.63	0.83	0.86	0.86	0.91	0.92
Spatial Distance	0.25	0.27	0.12	0.13	0.11	0.08	0.08
NMI	1.00	1.00	1.00	1.00	1.00	0.99	1.00
AIC	-109.34	-7.03	75.36	499.07	-194.65	-211.65	-438.54
BIC	-106.13	-3.54	79.57	505.17	-189.69	-208.34	-433.54
Data Size	184.00	242.00	498.00	3273.00	1052.00	202.00	1093.00
Training (s)	356.28	356.28	356.28	356.28	356.28	356.28	356.28
Testing (s)	102.49	103.27	102.98	91.94	104.72	163.13	116.37

Table A.9: Scenario A of Sampling Randomly

Error Metrics	Climate Csb	Climate Cwa	Climate Dsa	Climate Other	Climate Csa	Climate Cfa
Mean Target	3.75	3.75	3.75	3.75	3.75	3.75
F1	0.00001	0.95	0.00002	0.95	0.96	0.94
F2	0.00001	0.95	0.00005	0.93	0.95	0.96
F05	0.00001	0.94	0.0000125	0.97	0.96	0.93
Precision	0.00001	0.94	0.00001	0.98	0.97	0.93
Recall	0.00001	0.96	0.94	0.92	0.95	0.96
R2	0.20	0.33	0.40	0.57	0.63	0.69
Adjusted R2	1.95	3.55	-1.52	0.48	0.62	0.68
RMSE	0.85	1.38	1.26	1.21	1.18	1.18
MSE	0.72	1.91	1.59	1.46	1.40	1.38
MAE	0.66	0.99	1.02	0.90	0.89	0.91
MAPE	29.13	32.69	51.86	35.67	29.55	33.07
Accuracy	70.87	67.31	48.14	64.33	70.45	66.93
Pearson C.C.	0.46	0.67	0.66	0.78	0.82	0.87
Spearman C.C.	0.53	0.70	0.48	0.75	0.81	0.87
Spatial Distance	0.54	0.33	0.34	0.22	0.18	0.13
NMI	1.00	1.00	1.00	0.99	1.00	1.00
AIC	-7.02	27.29	31.53	98.55	384.40	398.88
BIC	-5.72	28.95	33.69	102.09	389.44	404.00
Data Size	27.00	39.00	64.00	254.00	1141.00	1227.00
Training (s)	1740.67	1740.67	1740.67	1740.67	1740.67	1740.67
Testing (s)	3.21	3.28	3.10	3.72	3.61	3.45

Table A.10: Scenario A of Sampling By Climate

A.2.2 Scenario B

Error Metrics	Climate Dsa	Climate Other	Climate Csa	Union of Climates	Climate Cwa	Climate Csb	Climate Cfa
Mean Target	2.57	3.49	3.99	3.74	4.51	2.28	4.00
F1	0.00002	0.92	0.95	0.95	0.97	0.00001	0.95
F2	0.00005	0.92	0.94	0.95	0.97	0.00001	0.95
F05	0.0000125	0.92	0.96	0.95	0.97	0.00001	0.96
Precision	0.00001	0.92	0.97	0.95	0.97	0.00001	0.96
Recall	0.92	0.92	0.94	0.95	0.98	0.00	0.95
R2	0.32	0.54	0.61	0.65	0.65	0.69	0.72
Adjusted R2	0.27	0.52	0.60	0.65	0.62	0.66	0.72
RMSE	1.14	1.34	1.28	1.18	0.82	0.54	1.10
MSE	1.30	1.78	1.63	1.38	0.67	0.29	1.22
MAE	0.86	0.98	0.95	0.85	0.57	0.41	0.81
MAPE	44.43	37.60	31.29	30.15	14.92	20.42	27.74
Accuracy	55.57	62.40	68.71	69.85	85.08	79.58	72.26
Pearson C.C.	0.63	0.76	0.80	0.83	0.83	0.85	0.86
Spearman C.C.	0.62	0.75	0.79	0.83	0.80	0.86	0.87
Spatial Distance	0.37	0.24	0.20	0.17	0.17	0.15	0.14
NMI	0.98	1.00	0.99	0.99	1.00	0.97	0.98
AIC	64.87	289.85	514.18	1062.60	-78.66	-225.55	217.42
BIC	68.36	294.06	519.14	1068.69	-75.35	-222.34	222.42
Data Size	242.00	498.00	1052.00	3273.00	202.00	184.00	1093.00
Training (s)	382.68	382.68	382.68	382.68	382.68	382.68	382.68
Testing (s)	109.85	109.28	121.44	96.07	110.75	109.40	110.12

Table A.11: Scenario B of Sampling Randomly

Error Metrics	Climate Csb	Climate Cwa	Climate Dsa	Climate Other	Climate Csa	Climate Cfa
Mean Target	3.75	3.75	3.75	3.75	3.75	3.75
F1	0.00001	0.95	0.00002	0.95	0.96	0.94
F2	0.00001	0.95	0.00005	0.93	0.95	0.96
F05	0.00001	0.94	0.0000125	0.97	0.96	0.93
Precision	0.00001	0.94	0.00001	0.98	0.97	0.93
Recall	0.00001	0.96	0.94	0.92	0.95	0.96
R2	0.20	0.33	0.40	0.57	0.63	0.69
Adjusted R2	1.95	3.55	-1.52	0.48	0.62	0.68
RMSE	0.85	1.38	1.26	1.21	1.18	1.18
MSE	0.72	1.91	1.59	1.46	1.40	1.38
MAE	0.66	0.99	1.02	0.90	0.89	0.91
MAPE	29.13	32.69	51.86	35.67	29.55	33.07
Accuracy	70.87	67.31	48.14	64.33	70.45	66.93
Pearson C.C.	0.46	0.67	0.66	0.78	0.82	0.87
Spearman C.C.	0.53	0.70	0.48	0.75	0.81	0.87
Spatial Distance	0.54	0.33	0.34	0.22	0.18	0.13
NMI	1.00	1.00	1.00	0.99	1.00	1.00
AIC	-7.02	27.29	31.53	98.55	384.40	398.88
BIC	-5.72	28.95	33.69	102.09	389.44	404.00
Data Size	27.00	39.00	64.00	254.00	1141.00	1227.00
Training (s)	1740.67	1740.67	1740.67	1740.67	1740.67	1740.67
Testing (s)	3.21	3.28	3.10	3.72	3.61	3.45

Table A.12: Scenario B of Sampling By Climate

A.2.3 Scenario C

Error Metrics	Climate Other	Climate Csa	Union of Climates	Climate Dsa	Climate Csb	Climate Cfa	Climate Cwa
Mean Target	3.51	4.01	3.75	2.57	2.28	4.01	4.52
F1	0.96	0.95	0.96	0.97	0.0001	0.97	0.98
F2	0.96	0.96	0.96	0.95	0.0001	0.97	0.98
F05	0.96	0.95	0.97	0.99	0.0001	0.97	0.99
Precision	0.96	0.94	0.97	1.01	0.0001	0.97	0.99
Recall	0.96	0.97	0.96	0.94	0.0001	0.97	0.98
R2	0.72	0.74	0.79	0.63	0.52	0.86	0.88
Adjusted R2	0.70	0.73	0.79	0.57	0.40	0.85	0.85
RMSE	0.98	0.99	0.91	0.86	0.64	0.78	0.51
MSE	0.96	0.99	0.83	0.74	0.41	0.60	0.26
MAE	0.76	0.75	0.67	0.64	0.52	0.58	0.39
MAPE	29.94	25.16	23.38	29.51	26.77	19.09	9.60
Accuracy	70.06	74.84	76.62	70.49	73.23	80.91	90.40
Pearson C.C.	0.86	0.88	0.89	0.80	0.82	0.93	0.94
Spearman C.C.	0.83	0.86	0.89	0.67	0.81	0.93	0.94
Spatial Distance	0.14	0.12	0.11	0.20	0.18	0.07	0.06
NMI	0.99	0.99	1.00	1.00	1.00	1.00	1.00
AIC	-20.67	-13.85	-623.90	-70.78	-161.91	-552.72	-272.38
BIC	-16.46	-8.90	-617.80	-67.29	-158.70	-547.73	-269.07
Data Size	498.00	1052.00	3273.00	242.00	184.00	1093.00	202.00
Training (s)	600.86	1016.22	1665.25	401.54	373.97	950.78	486.32
Testing (s)	104.39	115.59	97.16	112.07	107.27	109.31	110.07

Table A.13: Scenario C of Sampling Randomly

Error Metrics	Climate Csb	Climate Cwa	Climate Dsa	Climate Other	Climate Csa	Climate Cfa
F1	0.00001	0.97	0.96	0.95	0.95	0.95
F2	0.00001	0.97	0.98	0.95	0.95	0.96
F05	0.00001	0.97	0.95	0.94	0.96	0.94
Precision	0.00001	0.96	0.93	0.93	0.96	0.93
Recall	0.00001	0.97	1.00	0.96	0.95	0.97
Average	3.75	3.75	3.75	3.75	3.75	3.75
R2	0.65	0.70	0.67	0.52	0.65	0.75
Adjusted R2	2.10	-0.87	-0.05	0.43	0.63	0.74
RMSE	0.52	0.81	0.79	1.28	1.18	1.04
MSE	0.27	0.66	0.62	1.64	1.40	1.09
MAE	0.42	0.62	0.66	0.97	0.86	0.81
MAPE	22.00	17.17	34.69	41.09	29.25	29.24
Accuracy	78.00	82.83	65.31	58.91	70.75	70.76
Pearson C.C.	0.85	0.84	0.88	0.78	0.83	0.90
Spearman C.C.	0.86	0.85	0.62	0.69	0.81	0.89
Spatial Distance	0.15	0.16	0.12	0.22	0.17	0.10
NMI	0.97	1.00	0.98	1.00	1.00	0.99
AIC	-35.86	-16.45	-24.58	124.20	369.73	103.22
BIC	-34.49	-14.64	-22.57	127.71	374.74	108.30
Training (s)	1786.57	1786.57	1786.57	1786.57	1786.57	1786.57
Testing (s)	3.15	3.88	3.31	3.91	3.65	3.30

Table A.14: Scenario C of Sampling By Climate

A.2.4 Scenario D

Error Metrics	Climate Other	Climate Csb	Union of Climates	Climate Csa	Climate Cfa	Climate Dsa	Climate Cwa
Mean Target	3.58	2.30	3.74	3.99	4.01	2.47	4.48
F1	0.92	0.00002	0.92	0.92	0.95	0.97	0.98
F2	0.91	0.00001	0.90	0.91	0.91	0.96	0.97
F05	0.93	0.00005	0.94	0.92	0.99	0.97	0.98
Precision	0.94	0.89	0.96	0.92	1.02	0.97	0.99
Recall	0.91	0.00001	0.89	0.91	0.89	0.96	0.97
R2	0.22	0.25	0.35	0.42	0.48	0.53	0.74
Adjusted R2	0.19	0.18	0.34	0.41	0.47	0.50	0.72
RMSE	1.56	0.80	1.61	1.50	1.49	1.06	0.80
MSE	2.42	0.63	2.59	2.26	2.22	1.13	0.63
MAE	1.23	0.65	1.26	1.18	1.09	0.84	0.64
MAPE	53.04	36.21	47.22	38.57	32.43	41.02	17.51
Accuracy	46.96	63.79	52.78	61.43	67.57	58.98	82.49
Pearson C.C.	0.55	0.67	0.61	0.67	0.70	0.76	0.88
Spearman C.C.	0.57	0.61	0.63	0.63	0.72	0.66	0.88
Spatial Distance	0.45	0.33	0.39	0.33	0.30	0.24	0.12
NMI	0.99	0.98	1.00	1.00	1.00	0.99	1.00
AIC	443.07	-82.20	3117.54	859.56	874.68	32.14	-90.63
BIC	447.28	-78.99	3123.63	864.52	879.68	35.63	-87.33
Data Size	498.00	184.00	3273.00	1052.00	1093.00	242.00	202.00
Training (s)	811.40	336.71	1623.79	932.81	789.63	363.74	369.92
Testing (s)	1391.07	104.53	117.04	123.48	116.12	104.42	110.97

Table A.15: Scenario D of Sampling Randomly

Error Metrics	Climate Csb	Climate Dsa	Climate Other	Climate Csa	Climate Cwa	Climate Cfa
Mean Target	3.75	3.75	3.75	3.75	3.75	3.75
F1	0.00001	0.00001	0.88	0.91	0.96	0.95
F2	0.00001	0.00001	0.86	0.88	0.97	0.93
F05	0.00001	0.00001	0.91	0.93	0.95	0.97
Precision	0.00001	0.00001	0.92	0.95	0.94	0.99
Recall	0.00001	0.00001	0.84	0.87	0.98	0.91
R2	-0.64	-1.22	0.23	0.22	0.24	0.43
Adjusted R2	-3.01	-1.92	0.18	0.21	-0.17	0.42
RMSE	0.97	1.92	1.77	1.71	1.16	1.55
MSE	0.93	3.69	3.14	2.92	1.34	2.41
MAE	0.81	1.69	1.37	1.35	0.91	1.24
MAPE	43.38	103.21	56.67	43.41	27.29	46.89
Accuracy	56.62	NA	43.33	56.59	72.71	53.11
Pearson C.C.	0.48	0.59	0.48	0.47	0.75	0.70
Spearman C.C.	0.37	0.58	0.50	0.46	0.74	0.72
Spatial Distance	0.52	0.41	0.52	0.53	0.25	0.30
NMI	1.00	1.00	1.00	1.00	1.00	1.00
AIC	0.09	90.85	298.48	1228.51	15.60	1056.73
BIC	1.42	93.07	302.04	1233.55	17.45	1061.82
Data Size	28.00	68.00	259.00	1145.00	47.00	1200.00
Training (s)	1780.98	1788.26	1668.84	1788.26	1668.84	1788.26
Testing (s)	2.69	2.80	2.45	2.55	2.44	3.10

Table A.16: Scenario D of Sampling By Climate

Appendix B

Probabilistic and Uncertainty Experiments

This chapter includes all of the experimental results that were done but not mentioned in the original report. We have performed our experiments on all of our base (MC Dropout) and best (Deep Ensemble) probabilistic and uncertainty models across all the feature selection Scenarios (A,B,C, and D) mentioned in **Table [15.2]**. This chapter includes the following:

1. Experiments performed using MC Dropout model on all the data set and the clusters in **Section [B.1]**
2. Experiments performed using Deep Ensemble model on all the data set and the clusters in **Section [B.2]**

B.1 MCDropout Probabilistic Results

B.1.1 Scenario C

Error Metrics	Validation Union of Clusters	Validation Std	Testing Union of Clusters	Validation Cfa	Validation Std	Testing Cfa
Mean Target	3.75	-	3.75	3.99	-	3.99
F1	0.95	0.0010	0.96	0.95	0.005	0.95
F2	0.93	0.0010	0.93	0.91	0.01	0.94
F05	0.98	0.0015	0.98	0.99	0.01	0.96
Precision	1.00	0.0020	1.00	1.03	0.01	0.97
Recall	0.91	0.0013	0.92	0.88	0.01	0.93
R2	0.62	0.01	0.66	0.64	0.01	0.68
Adjusted R2	0.62	0.01	0.65	0.32	0.03	0.67
RMSE	1.24	0.01	1.17	1.52	0.03	1.17
MSE	1.54	0.02	1.37	2.31	0.09	1.36
MAE	0.86	0.01	0.83	0.95	0.03	0.83
MAPE	27.75	0.19	27.27	33.07	0.70	25.07
Accuracy	72.25	0.19	72.73	66.93	0.70	74.93
Pearson C.C.	0.79	0.00	0.82	0.81	0.01	0.83
Spearman C.C.	0.81	0.00	0.83	0.84	0.01	0.84
Spatial Distance	0.21	0.00	0.18	0.19	0.01	0.17
NMI	1.00	0.00	1.00	1.00	0.00	1.00
AIC	945.07	-	688.44	69.08	-	337.11
BIC	950.76	-	694.13	71.46	-	342.11
Data Size	10911	-	10911	3642	-	3642
Training (s)		-	0.00	6751.83	-	6751.83
Testing (s)	0.77	-	0.67	0.05	-	0.37

Table B.1: Scenario C of Clustering By Climate - Part 1

Error Metrics	Testing Csa	Validation Csa	Validation Std	Validation Cwa	Validation Std	Testing Cwa
Mean Target	4.00	4.00	-	4.49	-	4.49
F1	0.95	0.95	0.01	0.00002	0.47	0.96
F2	0.93	0.90	0.01	0.00005	0.47	0.96
F05	0.98	1.02	0.01	0.00001	0.46	0.96
Precision	1.00	1.07	0.02	0.00001	0.46	0.96
Recall	0.91	0.86	0.01	0.96	0.01	0.96
R2	0.61	0.57	0.04	0.83	0.03	0.54
Adjusted R2	0.60	0.12	0.08	1.09	0.02	0.44
RMSE	1.26	1.39	0.06	0.65	0.06	0.95
MSE	1.58	1.94	0.16	0.42	0.08	0.89
MAE	0.87	0.92	0.03	0.51	0.03	0.66
MAPE	24.67	23.06	0.87	11.86	0.96	16.52
Accuracy	75.33	76.94	0.87	88.14	0.96	83.48
Pearson C.C.	0.79	0.80	0.03	0.92	0.02	0.74
Spearman C.C.	0.81	0.83	0.02	0.86	0.03	0.75
Spatial Distance	0.21	0.20	0.03	0.08	0.02	0.26
NMI	1.00	1.00	0.00	1.00	0.00	1.00
AIC	484.90	50.33	-	-10.08	-	-20.47
BIC	489.86	52.62	-	-9.44	-	-17.16
Data Size	3509	3509	-	674	-	674
Training (s)	0.00		-		-	0.00
Testing (s)	0.32	0.03	-	0.01	-	0.15

Table B.2: Scenario C of Clustering By Climate - Part 2

Error Metrics	Validation Dsa	Validation Std	Testing Dsa	Validation Csb	Validation Std	Testing Csb
Mean Target	2.53	-	2.53	2.30	-	2.30
F1	0.00001	0	0.00002	0.00001	0	0.00001
F2	0.00001	0	0.00005	0.00001	0	0.00001
F05	0.00001	0	0.00001	0.00001	0	0.00001
Precision	0.00001	0	0.00001	0.00001	0	0.00001
Recall	0.00001	0	0.89	0.00001	0	0.00001
R2	-0.21	0.23	0.45	0.61	0.10	0.53
Adjusted R2	2.03	0.20	0.35	1.21	0.05	0.42
RMSE	0.76	0.07	1.14	0.59	0.07	0.64
MSE	0.58	0.11	1.29	0.34	0.09	0.41
MAE	0.60	0.06	0.78	0.44	0.07	0.49
MAPE	29.84	2.73	30.86	20.52	3.01	23.41
Accuracy	70.16	2.73	69.14	79.48	3.01	76.59
Pearson C.C.	0.56	0.06	0.70	0.79	0.06	0.74
Spearman C.C.	0.59	0.06	0.64	0.68	0.07	0.75
Spatial Distance	0.44	0.06	0.30	0.21	0.06	0.26
NMI	1.00	0.00	1.00	1.00	0.00	1.00
AIC	-7.90	-	64.15	-12.93	-	-162.32
BIC	-7.01	-	67.63	-12.29	-	-159.11
Data Size	807	-	807	614	-	614
Training (s)	155.18	-	155.18	136.40	-	136.40
Testing (s)	0.01	-	0.15	0.01	-	0.05

Table B.3: Scenario C of Clustering By Climate - Part 3

B.2 Deep Ensemble Probabilistic Results

B.2.1 Scenario A

Error Metrics	Union of Climates Validation	Union of Climates Validation Std	Union of Climate Testing
Mean Target	3.75	-	3.75
F1	0.94	0.003	0.95
F2	0.92	0.01	0.93
F05	0.97	0.01	0.96
Precision	0.99	0.01	0.98
Recall	0.90	0.01	0.92
R2	0.67	0.02	0.66
Adjusted R2	0.59	0.02	0.65
RMSE	1.28	0.03	1.18
MSE	1.65	0.08	1.39
MAE	0.86	0.01	0.82
MAPE	29.09	1.00	25.87
Accuracy	70.91	1.00	74.13
Pearson C.C.	0.83	0.01	0.81
Spearman C.C.	0.85	0.01	0.83
Spatial Distance	0.17	0.01	0.19
NMI	1.00	0.00	1.00
AIC	119.40	-	1069.42
BIC	122.86	-	1075.51
Data Size	10911.00	-	10911.00
Training (s)	505.57	-	505.57
Testing (s)	0.08	-	1.03

Table B.4: Scenario A of Union of Climates

B.2.2 Scenario B

Error Metrics	Union of Climates Validation	Union of Climates Validation Std	Union of Climate Testing
Mean Target	3.75	-	3.75
F1	0.95	0.004	0.95
F2	0.91	0.01	0.92
F05	0.99	0.004	0.97
Precision	1.02	0.005	0.98
Recall	0.89	0.01	0.91
R2	0.62	0.01	0.59
Adjusted R2	0.59	0.01	0.58
RMSE	1.38	0.02	1.29
MSE	1.90	0.06	1.66
MAE	0.91	0.01	0.90
MAPE	30.41	0.93	28.80
Accuracy	69.59	0.93	71.20
Pearson C.C.	0.80	0.01	0.77
Spearman C.C.	0.82	0.00	0.79
Spatial Distance	0.20	0.01	0.23
NMI	1.00	0.00	1.00
AIC	152.23	-	1666.63
BIC	155.69	-	1672.72
Data Size	10911.00	-	10911.00
Training (s)	470.68	-	470.68
Testing (s)	1.14	-	1.14

Table B.5: Scenario B of Union of Climates

B.2.3 Scenario C

Error Metrics	Union of Climates Validation	Union of Climates Validation Std	Union of Climate Testing
Mean Target	3.75	3.75	-
F1	0.95	0.95	0.002
F2	0.93	0.92	0.01
F05	0.97	0.99	0.01
Precision	0.98	1.02	0.01
Recall	0.92	0.90	0.01
R2	0.67	0.69	0.02
Adjusted R2	0.67	0.63	0.02
RMSE	1.15	1.26	0.03
MSE	1.33	1.58	0.08
MAE	0.80	0.82	0.02
MAPE	25.49	27.40	1.09
Accuracy	74.51	72.60	1.09
Pearson C.C.	0.82	0.84	0.01
Spearman C.C.	0.84	0.86	0.01
Spatial Distance	0.18	0.16	0.01
NMI	1.00	1.00	0.00
AIC	931.57	109.84	-
BIC	937.66	113.30	-
Data Size	10911.00	10911.00	-
Training (s)	511.57	-	511.57
Testing (s)	1.01	-	1.01

Table B.6: Scenario C of Union of Climates

Error Metrics	Validation Cfa	Testing Cfa	Validation Csa	Testing Csa	Validation Csb	Testing Csb	Validation Cwa	Testing Cwa	Validation Dsa	Testing Dsa	Validation Other	Testing Other
Mean Target	3.99	3.99	4.00	4.00	2.30	2.30	4.49	4.49	2.53	2.53	3.51	3.51
F1	0.95	0.95	0.95	0.95	0.00001	0.00001	1.00	0.96	0.00001	0.00002	0.96	0.93
F2	0.91	0.94	0.92	0.94	0.00001	0.00001	0.98	0.97	0.00001	0.00005	0.95	0.90
F05	0.99	0.97	1.00	0.97	0.00001	0.00001	1.01	0.96	0.00001	0.00001	0.98	0.97
Precision	1.03	0.97	1.03	0.98	0.00001	0.00001	1.03	0.95	0.00001	0.00001	0.99	0.99
Recall	0.88	0.93	0.89	0.92	0.00001	0.00001	0.96	0.97	0.00001	0.90	0.94	0.88
R2	0.67	0.69	0.67	0.65	0.62	0.60	0.83	0.59	0.44	0.48	0.65	0.50
Adjusted R2	0.38	0.68	0.33	0.64	1.21	0.50	1.09	0.50	2.22	0.39	7.13	0.46
RMSE	1.45	1.16	1.21	1.20	0.58	0.59	0.65	0.89	0.83	1.10	1.33	1.42
MSE	2.11	1.34	1.47	1.44	0.34	0.35	0.42	0.80	0.69	1.22	1.76	2.03
MAE	0.89	0.82	0.78	0.82	0.39	0.45	0.52	0.61	0.70	0.80	1.02	1.02
MAPE	29.28	24.50	20.59	23.81	18.63	21.15	12.51	15.71	35.03	33.83	40.15	35.65
Accuracy	70.72	75.50	79.41	76.19	81.37	78.85	87.49	84.29	64.97	66.17	59.85	64.35
Pearson C.C.	0.83	0.83	0.84	0.81	0.80	0.78	0.91	0.78	0.61	0.70	0.81	0.70
Spearman C.C.	0.86	0.85	0.87	0.82	0.65	0.78	0.87	0.77	0.64	0.64	0.76	0.71
Spatial Distance	0.17	0.17	0.16	0.19	0.20	0.22	0.09	0.22	0.39	0.30	0.19	0.30
NMI	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
AIC	61.70	325.17	30.28	383.97	-13.04	-190.15	-10.13	-43.72	-4.76	50.07	22.34	355.20
BIC	64.09	330.16	32.57	388.93	-12.41	-186.93	-9.49	-40.42	-3.87	53.56	23.92	359.41
Data Size	3642.00	3642.00	3509.00	3509.00	614.00	614.00	674.00	674.00	807.00	807.00	1665.00	1665.00
Training (s)	223.90	223.90	223.16	223.16	113.26	113.26	117.56	117.56	113.26	113.26	154.14	154.14
Testing (s)	0.03	0.35	0.04	0.30	0.01	0.06	0.01	0.05	0.01	0.07	0.04	0.15

Figure B.1: Scenario C of Clustering By Climate

B.2.4 Scenario D

Error Metrics	Union of Climates Validation	Union of Climates Validation Std	Union of Climate Testing
Mean Target	3.75	-	3.75
F1	0.92	0.01	0.93
F2	0.87	0.01	0.90
F05	0.97	0.01	0.97
Precision	1.01	0.02	0.99
Recall	0.84	0.01	0.88
R2	0.26	0.01	0.34
Adjusted R2	0.20	0.01	0.34
RMSE	1.93	0.01	1.63
MSE	3.74	0.06	2.66
MAE	1.41	0.01	1.24
MAPE	49.37	1.51	43.27
Accuracy	50.63	1.51	56.73
Pearson C.C.	0.51	0.01	0.58
Spearman C.C.	0.50	0.01	0.60
Spatial Distance	0.49	0.01	0.42
NMI	1.00	0.00	1.00
AIC	312.21	-	3197.24
BIC	315.67	-	3203.33
Data Size	10911.00	-	10911.00
Training (s)	511.57	-	511.57
Testing (s)	0.99	-	0.99

Table B.7: Scenario D of Union of Climates

References

Allen, P. L. H. T., R., & Jensen, M. (2019). Evapotranspiration information reporting: I. factors governing measurement accuracy.

Allen, R. G. (2005). *The asce standardized reference evapotranspiration equation.* American Society of Civil Engineers.

Allen, R. G., Pereira, L. S., Howell, T. A., & Jensen, M. E. (2011). Evapotranspiration information reporting: I. factors governing measurement accuracy. *Agricultural Water Management*, *98*(6), 899–920.

Allen, R. G., Walter, I. A., Elliott, R. L., Howell, T. A., Itenfisu, D., Jensen, M. E., & Snyder, R. L. (2005). The asce standardized reference evapotranspiration equation..

Ameriflux. (2019). *Search ameriflux sites.* Retrieved from `https://ameriflux.lbl.gov/sites/site-searc`

Baier, W., & Robertson, G. W. (1965). *Estimation of latent evaporation from simple weather observations.* Retrieved from `https://cdnsciencepub.com/doi/pdf/10.4141/cjps65-051`

Balaji Lakshminarayanan, A. P., & Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles.

Bishop, C. M. (1994). Mixture density networks.

Bloch, M. (2019, Jan). *Lecture 5 - linear discriminant analysis and logistic classification.* Retrieved from `https://bloch.ece.gatech.edu/ece6254sp19/lecture5-v3.pdf`

Branco, P., Torgo, & Ribeiro, R. P. (2017). Smogn: a pre-processing approach for imbalanced regression. In *First international workshop on learning with imbalanced domains: Theory and applications* (pp. 36–50).

Brownlee, J. (n.d.). *Framework for better deep learning.*

Brownlee, J. (2016). *How to check if time series data is stationary with python.* Retrieved from `https://machinelearningmastery.com/time-series-data-stationary-python/`

Brownlee, J. (2019, Aug). *Framework for better deep learning.* Retrieved from `https://machinelearningmastery.com/framework-for-better-deep-learning/`

Brownlee, J. (2020a, Aug). *How to calculate precision, recall, and f-measure for imbalanced classification.* Retrieved from `https://`

machinelearningmastery.com/precision-recall-and-f-measure-for-imbalanced-classification

Brownlee, J. (2020b, Aug). *Probabilistic model selection with aic, bic, and mdl.* Retrieved from https://machinelearningmastery.com/probabilistic-model-selection-measures/

Chapter 4 - determination of eto. (2019a). Retrieved from http://www.fao.org/3/X0490E/x0490e08.htm

Chapter 4 - determination of eto. (2019b). Retrieved from http://www.fao.org/3/X0490E/x0490e08.htm

Deshpande, M. (2019, Sep). *Clustering with gaussian mixture models.* Retrieved from https://pythonmachinelearning.pro/clustering-with-gaussian-mixture-models/

EEflux. (2019). *Eeflux-level1.appspot.com.* Retrieved from https://eeflux-level1.appspot.com/

EEFlux: A landsat-based evapotranspiration mapping tool on the google earth engine. (2015, nov). In *2015 ASABE / IA irrigation symposium: Emerging technologies for sustainable irrigation - a tribute to the career of terry howell, sr. conference proceedings.* American Society of Agricultural and Biological Engineers. Retrieved from https://doi.org/10.13031%2Firrig.20152143511 doi: 10.13031/irrig.20152143511

El-Nesr, D. M. (2019, Dec). *Filling gaps of a time-series using python.* Medium. Retrieved from https://medium.com/@drnesr/filling-gaps-of-a-time-series-using-python-d4bfddd8c460

Fan, J., Yue, W., Wu, L., Zhang, F., Cai, H., Wang, X., ... Xiang, Y. (2018). Evaluation of svm, elm and four tree-based ensemble models for predicting daily reference evapotranspiration using limited meteorological data in different climates of china. *Agricultural and Forest Meteorology*, *263*, 225–241.

Fan, W. X. L. X., J., & Xiang, Y. (2019). Evaluation of svm, elm and four tree-based ensemble models for predicting daily reference evapotranspiration using limited meteorological data in different climates of china.

Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*.

Foolad, F., Blankenau, P., Kilic, A., Allen, R. G., Huntington, J. L., Erickson, T. A., ... others (2018). Comparison of the automatically calibrated google evapotranspiration application—eeflux and the manually calibrated metric application.

Gal, Y., & Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning.

Granata, F. (2019). Evapotranspiration evaluation models based on machine learning algorithms—a comparative study. *Agricultural Water Management*, *217*, 303 - 315. Retrieved from http://www.sciencedirect.com/science/article/pii/S0378377418312800 doi: https://doi.org/

10.1016/j.agwat.2019.03.015

Gustafsson, F. K., Danelljan, M., & Schon, T. B. (2020). Evaluating scalable bayesian deep learning methods for robust computer vision. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition workshops* (pp. 318–319).

Gutiérrez, P. A., & Hervás-Martínez, C. (2011). Hybrid artificial neural networks: models, algorithms and data. In *International work-conference on artificial neural networks* (pp. 177–184).

Hansen, C. (2019, Nov). *Nested cross-validation python code.* Machine Learning From Scratch. Retrieved from `https://mlfromscratch.com/nested-cross-validation-python-code/`

Huang, G., Wu, L., Ma, X., Zhang, W., Fan, J., Yu, X., ... Zhou, H. (2019). Evaluation of catboost method for prediction of reference evapotranspiration in humid regions. *Journal of Hydrology*, *574*, 1029–1041.

Hullermeir, E., & Waegeman, W. (2020). Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. Retrieved from `https://arxiv.org/pdf/1910.09457.pdf`

Ilg, E., Cicek, O., Galesso, S., Klein, A., Makansi, O., Hutter, F., & Brox, T. (2018). Uncertainty estimates and multi-hypotheses networks for optical flow. In *Proceedings of the european conference on computer vision (eccv)* (pp. 652–667).

is currently a student at the University of California San Diego pursuing a PhD in Biostatistics., C. P. (2020, Apr). *Tutorial: Understanding linear regression and regression error metrics.* Retrieved from `https://www.dataquest.io/blog/understanding-regression-error-metrics/`

Jaafar, H. H., & Ahmad, F. A. (2020). Time series trends of landsat-based et using automated calibration in metric and sebal: The bekaa valley, lebanon. *Remote Sensing of Environment*, *238*, 111034.

Kana, M. (2019). Uncertainty in deep learning. how to measure? *Medium*. Retrieved from `https://towardsdatascience.com/my-deep-learning-model-says-sorry-i-dont-know-the-answer-that-s-absolutely-ok-50ffa562cb0b`

Kaneko, A., & Kennedy, T. (2019). *Deep learning for crop yield prediction in africa.* Retrieved from `https://aiforsocialgood.github.io/icml2019/accepted/track1/pdfs/20_aisg_icml2019.pdf`

Kiat, L. S. (2018). *maml-reptile.* Retrieved from `https://github.com/greentfrapp/maml-reptile`

Learning, M. (2019). *Machine learning from scratch.* Retrieved from `https://mlfromscratch.com/nested-cross-validation-python-code/`

Liakos, K. G., Busato, P., Moshou, D., Pearson, S., & Bochtis, D. (2018). Machine learning in agriculture: A review. *Sensors*, *18*(8), 2674.

Maklin, C. (2019, Jul). *K-means clustering python example.* Towards Data Science. Retrieved from `https://towardsdatascience.com/`

machine-learning-algorithms-part-9-k-means-example-in-python-f2ad05ed5203

Marco Tulio Ribeiro, S. S., & Guestrin, C. (2016). Local interpretable model-agnostic explanations (lime): An introduction. *oreilly*. Retrieved from https://www.oreilly.com/content/introduction-to-local-interpretable-model-agnostic-explanations-lime/

Mauder. (2013). *Considerations for the energy balance residual (ebr) correction.* Retrieved from https://www.licor.com/env/support/Tovi/topics/energy-balance-closure-correction.html#ConsiderationsfortheenergybalanceresidualEBRcorrection

Metric. (2019). Retrieved from https://eeflux-level1.appspot.com/

Mishra, D. (2019, Dec). *Regression: An explanation of regression metrics and what can go wrong.* Towards Data Science. Retrieved from https://towardsdatascience.com/regression-an-explanation-of-regression-metrics-and-what-can-go-wrong-a39a9793d914

Need for feature engineering in machine learning. (2019). Retrieved from https://towardsdatascience.com/need-for-feature-engineering-in-machine-learning-897df2ed00e

Nichol, A., Achiam, J., & Schulman, J. (2018). On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*.

Nichol, A., & Schulman, J. (2018). Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, *2*(3), 4.

Normalized mutual information estimating clustering quality. (2019). Retrieved from https://course.ccs.neu.edu/cs6140sp15/7_locality_cluster/Assignment-6/NMI.pdf

otoro.net. (2015). *Mixture density networks with tensorflow.* Retrieved from https://blog.otoro.net/2015/11/24/mixture-density-networks-with-tensorflow/

Parsa, & Movahedi. (2020). Toward safer highways, application of xgboost and shap for real-time accident detection and feature analysis.

Parsa, A. B., & Movahedi, A. (2020). *Toward safer highways, application of xgboost and shap for real-time accident detection and feature analysis.* Retrieved from https://pubmed.ncbi.nlm.nih.gov/31864931/

Rank correlation: Spearman coefficient, methods, formula, examples. (2019, Dec). Retrieved from https://www.toppr.com/guides/business-mathematics-and-statistics/correlation-and-regression/rank-correlation

Reference evapotranspiration (etref). (2019). Retrieved from https://www.weap21.org/WebHelp/Mabia_Alg_ETRef.htm

research interests lies in the field of Machine Learning, P. S., an enthusiasm for learning new skills, D. L. P., & technologies. (2020, Apr). *A beginner's guide to hierarchical clustering and how to perform it in python.* Retrieved from https://www.analyticsvidhya.com/blog/2019/05/beginners-guide-hierarchical-clustering/

Ribeiro, & Torgo. (2008). Utility-based performance measures for regression. In *Proceedings of the 3rd workshop on evaluation methods for machine learning, in conjunction with the 25th international conference on machine learning (icml 2008).*

Search ameriflux sites. (2019, Jul). Retrieved from `https://ameriflux.lbl.gov/sites/site-search`

sklearn.metrics.normalized_mutual_info_score. (2019). Retrieved from `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.normalized_mutual_info_score.html`

Sungjoon Choi, S. L., Kyungjae Lee, & Oh, S. (2017). Uncertainty-aware learning from demonstration using mixture density networks with sampling-free variance modeling.

Wateraccounting. (2019). *wateraccounting/sebal.* Retrieved from `https://github.com/wateraccounting/SEBAL`

Wu, L., & Fan, J. (2019). Comparison of neuron-based, kernel-based, tree-based and curve-based machine learning models for predicting daily reference evapotranspiration. *PloS one*, *14*(5), e0217520.

Yin, W. (2020). Meta-learning for few-shot natural language processing: A survey. *arXiv preprint arXiv:2007.09604v1*.

"why should i trust you?" explaining the predictions of any classifier. (1026). Retrieved from `https://arxiv.org/pdf/1602.04938.pdf`