2102531 System Identification

Photovaltaic System Modeling
Semester 1/2018

Kitinan Boonravee
Jeerapat Jitnuant
Jitkomut Songsiri
Department of Electrical Engineering
Faculty of Engineering
Chulalongkorn University

November 26, 2018

Abstract

Nowadays, our world has limited energy resource. The use of renewable energy can not be avoided. In order to develop management of energy resource to be systematic, the use of renewable energy such as Solar cells is necessary. This term project is going to proposed the new model for estimate power prediction of Photovoltaic model (PV model) with cell temperature and irradiance as the inputs. The proposed model is going to compare with Polynomial model and Artificial Neural Networks (ANN) model in terms of Root Mean-square Error (RMSE).

1 Introduction

Photovoltaic (PV) is the semiconductor device used to convert solar irradiance into electrical power. Another factor that affected to the converting efficiency is a temperature on photovoltaic device, it is a challenge to estimate the electrical power which has many affected variables. In previous research, The PV can be modeled as the simplified equivalent circuit [3] on a single diode model which can be explained the characteristic of PV by the nonlinear equation on the solar irradiance and temperature. Similar to [6] that proposed a method to find the parameters of the nonlinear I–V equation for the single-diode PV model by optimization method. The simplified relation on generated power in solar irradiance and temperature is proposed in the polynomial models [2] which are alternative method of power estimation. Other techniques used are the Artificial Neural Network (ANN) and Support Vector Regression (SVR) [4] which can be applied to explain the nonlinear relation between the electrical power and Irradiance and between the electrical power and temperature.

Moreover, the transient response in power inverter of the PV system is caused by the capacitor circuit components in the system. However, the equivalent circuit [3, 6] and polynomial models [2] were not considered the solar irradiance and temperature data in the previous time. Thus, the estimation of the generated power in PV model can be improved by using the previous solar irradiance and the previous temperature.

This term project aims to improve the generated power estimation by using ANN and SVR models. The ANN model with lagging input will be compared the fitting performance with non-lagging input, SVR model, and polynomial models in terms of Root Mean Squared Error (RMSE). The methods and models that used to refine estimates generated power from the PV system in this work are described in Section 4. The estimation results of each models in this work are summarized in Section 5. Finally, conclusion is presented in Section 6.

2 Backgrounds

This section will describe about background of photovoltaic model based on the ideal equivalent circuit. The power generated (P) is the function of the solar irradiance (I) and cell temperature (T) which are explained by using the model in this section.

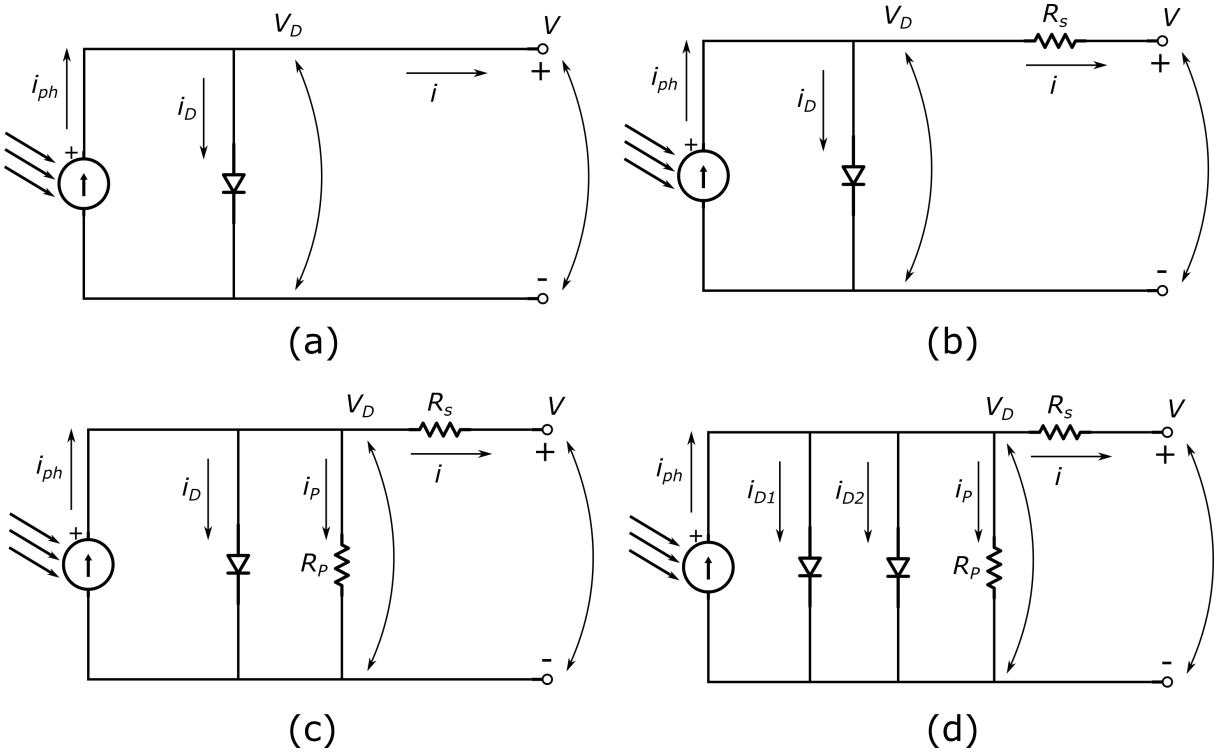


Figure 1: Equivalent PV cell electrical circuits:(a) ideal model; (b) one-diode only with R_s; (c) one-diode with R_s and R_p and (d) two-diode models.

2.1 Equivalent electrical circuit of photovoltaic module

The equation which descibes the relationship between the current (i) and voltage (V) of the PV model in Fig.1 (a, b), (c), and (d) can be derived as equation (1), (2), and (3) respectively.

$$i = i_\mathrm{ph} - i_\mathrm{D}, \tag{1}$$
$$i = i_\mathrm{ph} - i_\mathrm{D} - i_\mathrm{p}, \tag{2}$$
$$i = i_\mathrm{ph} - i_\mathrm{D1} - i_\mathrm{D2} - i_\mathrm{p}. \tag{3}$$

where i_ph is the photo current(A). The relation of diode current (i_D) on voltage (v) and diode thermal voltage (V_t) is expressed as,

$$i_\mathrm{D} = i_0(e^{V_\mathrm{D}/V_\mathrm{t}} - 1) \tag{4}$$

where the junction thermal voltage (V_{ti}) of the ith diode is $V_{ti} = A_i k_\mathrm{b} T_\mathrm{c}/q$, A_i is the ideality factor of the ith diode, $k_\mathrm{b} = 1.38 \times 10^{-23}$ (Joule/Kelvin) is the Boltzmann constant, $q = 1.6 \times 10^{-19}$ (Coulomb) is the elementary charge, and T_c is the cell temperature in Kelvin.

The diode saturation current (i_0) can be expressed as a function of temperature from [3], as follows:

$$i_0 = C_0 T_\mathrm{c}^3 \cdot \exp\left(-\frac{E_\mathrm{g}}{k_\mathrm{b} T_\mathrm{c}}\right). \tag{5}$$

where E_g is the band gap energy of the semiconductor and C_0 is a constant depending on material parameters and can be expressed by

$$C_0 = \frac{i_{0,\mathrm{stc}}}{T_\mathrm{c}^3 \exp(-E_\mathrm{g}/k_\mathrm{b}T_{\mathrm{c,stc}})}. \tag{6}$$

According to [1], $i_{0,\mathrm{stc}}$ and A_stc are derived from physics and can be expressed as follows:

$$i_{0,\mathrm{stc}} = \left(i_{\mathrm{sc,stc}} - \frac{v_{\mathrm{oc,stc}}}{R_{\mathrm{p,stc}}}\right) \cdot \exp\left(-\frac{v_{\mathrm{oc,stc}}}{A_\mathrm{stc}}\right). \tag{7}$$

$$A_\mathrm{stc} = \frac{v_{\mathrm{mpp,stc}} + i_{\mathrm{mpp,stc}} \cdot R_{\mathrm{s,stc}} + v_{\mathrm{oc,stc}}}{\ln\left(i_{\mathrm{sc,stc}} - \frac{v_{\mathrm{mpp,stc}}}{R_{\mathrm{p,stc}}} - i_{\mathrm{mpp,stc}}\right) - \ln\left(i_{\mathrm{sc,stc}} - \frac{v_{\mathrm{oc,stc}}}{R_{\mathrm{p,stc}}}\right) + \left(\frac{R_{\mathrm{p,stc}} \cdot i_{\mathrm{mpp,stc}}}{R_{\mathrm{p,stc}} \cdot i_{\mathrm{sc,stc}} - v_{\mathrm{oc,stc}}}\right)}. \tag{8}$$

where $i_{\mathrm{sc,stc}}$ is the short-circuit current, $v_{\mathrm{oc,stc}}$ is the open-circuit voltage, $v_{\mathrm{mpp,stc}}$ is the voltage at maximum power, $i_{\mathrm{mpp,stc}}$ is the current at maximum power. There parameters are received from manufactureer's datasheet. When $R_{\mathrm{p,stc}}$ is respectively the slope of the I–V curve in short circuit points.

The photocurrent i_ph is highly depends on solar irradiance. According to [6], the equation can be derived from term of irradiance (I) and cell temperature (T_c) as follow:

$$i_\mathrm{ph} = (i_{\mathrm{ph,stc}} + K_\mathrm{I}(T_\mathrm{C} - T_{\mathrm{C,stc}}))\frac{I}{I_\mathrm{stc}}. \tag{9}$$

where $i_{\mathrm{ph,stc}}$ is the light-generated current at the nominal condition (usually 25C° and 1000W/m^2), (T_C and $T_{\mathrm{C,stc}}$ being the cell and nominal temperatures [in Kelvin], respectively, I (W/m^2) is the irradiation on the device surface, and I_stc is the nominal irradiation.

Substitute (4) - (9) into (2) the final equation of the model can then be expressed by:

$$\begin{aligned} i = {} & i_{\mathrm{ph,stc}} + K_I(T_\mathrm{C} - T_{\mathrm{C,stc}}))\frac{I}{I_\mathrm{stc}} \\ & - \left(i_{\mathrm{sc,stc}} - \frac{v_{\mathrm{oc,stc}}}{R_{\mathrm{p,stc}}}\right) \cdot \exp\left(-\frac{v_{\mathrm{oc,stc}}}{A_\mathrm{stc}}\right)(e^{(V+iR_\mathrm{s})/V_\mathrm{t}} - 1) - \frac{V + iR_\mathrm{s}}{R_\mathrm{p}} \end{aligned} \tag{10}$$

An another proposed simplified model is described by considering at maximum power generated (V_m, i_m) as

$$V_\mathrm{m} = A\log(\frac{i_\mathrm{ph} - i_\mathrm{m}}{i_0}) \tag{11}$$

Thus, the maximum power output can be expressed as:

$$P_\mathrm{m} = V_\mathrm{m} i_\mathrm{m} \tag{12}$$

The simplified maximum power output model as a function of irradiance and temperature can be expressed as follows:

$$\begin{aligned} P_m = {} & \left(c_1 IT + c_2 T^2 + c_3 T\right) \cdot \log\left(c_4 \frac{I}{T^3} + c_5 \frac{I}{T^2} + \frac{c_6}{T^2} + \frac{c_7}{T^3}\right) \\ & + \left(c_8 I^2 + c_9 T^2 + c_{10} IT + c_{11} I + c_{12} T + c_{13}\right). \end{aligned} \tag{13}$$

where $c_i, c_2, ..., c_{13}$ are the model coefficients.

2.2 The polynomial model

O. GERGAUD (2002) [3] has simplified the PV models into the polynomial models, as follow:

$$P = k_1 I[1 + k_2(T - T_{ref})]. \tag{14}$$

The estimation error when maximum power P be zero but irradiance I is not zero, may improve fitting result by adding a parameter (K_3) to the equation (14) as:

$$P = k_1(k_3 + I)[1 + k_2(T - T_{ref})]. \tag{15}$$

These simplified model has made it possible to estimate the maximum power supplied by the solar cell system.

2.3 Artificial neural network model

Artificial Neural Networks is a model which simulate from human brain. It is also called "blackbox model" because, it can learn from features(input) by itself like. As you can see in Fig.2, each nodes represent a neuron or activation function from input and layers by layers. Each nodes are connected by grey lines which represent weight to create linear equation as the input of activation function in each nodes in next layer.

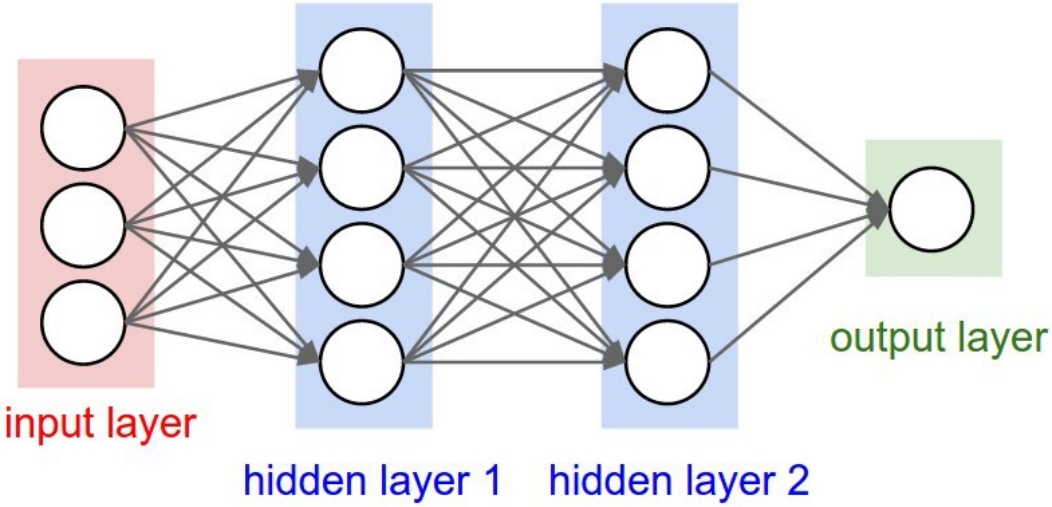


Figure 2: Artificial Neural Networks Model.

2.4 Support Vector Regression (SVR)

The support vector regression (SVR) are very specific class of algorithms, characterized by usage of kernels, absence of local minima, sparseness of the solution and capacity control obtained by acting on the margin, or on number of support vectors, etc. The estimator function can be described in the case of linear functions as

$$f(x) = \langle w, x_i \rangle + b \tag{16}$$

where $\langle \cdot, \cdot \rangle$ denotes the dot product. According to the the research papers [5]. Support Vector Regression used to construct the hyperplane and create margin. We need error to be less than epsilon and we need C to trade-off between flatness and the amount of deviation. Our objective function is described as below:

$$\underset{w,b}{\text{minimize}} \frac{1}{2}\|w\|^2 \ + C\sum_{i=1}^{\ell}(\xi_i + \xi_i^*) \tag{17}$$

subject to

$$\begin{gathered} y_i - \langle w, x_i \rangle - b \leq \varepsilon + \xi_i \\ \langle w, x_i \rangle + b - y_i \leq \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \end{gathered}$$

where

- $y \in \mathbb{R}^{\ell}$ is the target label.
- $x \in \mathbb{R}^{\ell}$ is the feature.
- $\|w\|$ is the norm of normal vector perpendicular to hyperplane.
- b is the bias term.
- C is used to trade-off between $\|w\|$ and the loss which unacceptable.
- ξ is a slack variables use to measure loss.
- ε is the constant deviation from hyperplane to margin.

The constant $C > 0$ determines the trade-off between the flatness of f and the amount up to which deviations larger than ε are tolerated. This corresponds to dealing with a so called ε-insensitive loss function $|\xi|_{\varepsilon}$ described by

$$|\xi|_{\varepsilon} := \begin{cases} 0, & \text{if} |\xi| \leq \varepsilon \\ |\xi| - \varepsilon, & \text{Otherwise} \end{cases}$$

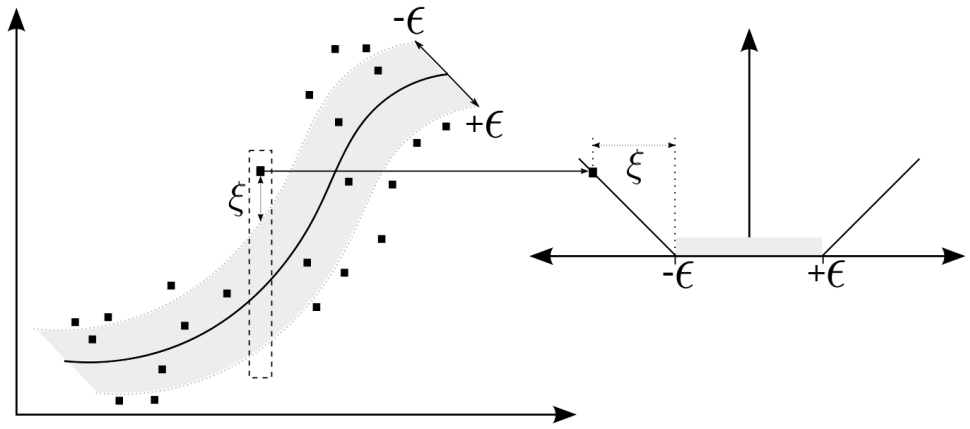


Figure 3: The soft margin loss setting for a linear SVM.

As we can see from nonlinear relation between power generated and solar irradiance and cell temperature from the derivation of PV equivalent circuit model We use Kernel function to map input space to feature space. The important kernel function is described as below:

- **linear**: k(x,x') = $\langle x, x' \rangle$.
- **polynomial**: k(x,x') =$\gamma(\langle x, x' \rangle + r)^d$.
- **rbf**: k(x,x') =exp$(-\gamma \|x - x'\|^2)$.
- **sigmoid**: k(x,x') =tanh$(\gamma \langle x, x' \rangle + r)$.

where

- $x, x' \in \mathbb{R}^{\ell}$ is the vector in input space.
- d is the degree of kernel polynomial function.
- γ is the kernel coefficient.
- r is the constant term.

2.5 Prediction performance index and statistics formulas

Root Mean Square Error (RMSE) is a frequently used measure of the differences between values (sample or population values) predicted by a model or an estimator and the values observed.

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{k=1}^{N}\left(x(k) - \hat{x}(k)\right)^2} \tag{18}$$

where N is the number of sampling, k is the sampling index, $\hat{x}(k)$ is the predicted values, and $x(k)$ is the measurement values.

Standard deviation (SD, σ) is a measure that used to quantify the amount of variation or dispersion of a set of data values.

$$\sigma = \sqrt{\frac{1}{N}\sum_{k=1}^{N}\left(x(k) - \bar{x}\right)^2} \tag{19}$$

where N is the number of sampling, k is the sampling index, and $\bar{x}(k)$ is the mean values of measurement values $x(k)$.

3 Problem statement

In this term project, Our goal is to develop the models for the maximum power generated estimation by using the solar irradiance and cell temperature values which are collected from the solar site at EE building, Chulalongkorn university. The models are included Unbiased-polynomial model, Biased-polynomial model, Artificial Neural Networks(ANN) with Lagging model,Artificial Neural Networks(ANN) with Non-Lagging model and Support Vector Regression (SVR) model. The fitting performance of each model will be compared by the term of root mean-square error (RMSE).

4 Methodology

In this section, we will demonstrate the methodology for solving the model parameters to estimate the maximum power generated. The procedure is consist of three major steps which is data preprocessing, determination of model parameters, and model validation.

4.1 Data preprocessing

4.1.1 Data set

We begin to experiment in two scenarios which are uncleaned dataset and cleaned dataset separately.For uncleaned dataset, we randomly shuffle by day and split the measurement data into training set and validation set. For cleaned dataset, we delete outliers from the measurement data and randomly shuffle by day. Then, we split the measurement data into training set and validation set. Finally, we have two training set from cleaned and uncleaned dataset, and two validation set from cleaned and uncleaned dataset.

4.1.2 Outliers

Consider the undesired measurement data of power generated ($P \in \mathbb{R}^N$) and solar irradiance ($I \in \mathbb{R}^N$) is might be problematic to estimate. In this section, we proposed the method to remove the outliers of power generated and solar irradiance data. The key idea to identify the outliers data by comparing the proportion of power and solar irradiance, if the proportion of both data is too large, we can conclude that is an outlier in measurements. To identify the outliers data, we

simplified by normalizing the solar irradiance data to be same ratio with power data and measure the difference between the power and normalized solar irradiance data as

$$y(k) = \frac{P(k)}{\bar{P}} - \frac{I(k)}{\bar{I}}. \qquad \text{For} \quad k = 1, 2, ..., N \tag{20}$$

where k is the index of measurement data, N is the number measurement data, $y \in \mathbb{R}^N$ is the residue value, $\bar{P}$ and $\bar{I}$ are the sampling mean values of power and solar irradiance data respectively. Then, we using the Z-Score of y from (19) to be the criterion for filler out these outliers.

- If $-3 \leq \frac{y(k)-\bar{y}}{\sigma} \leq 3 : y(k)$ is not an outlier data. For $k = 1, 2, ..., N$

where k is the sampling index, $\bar{y}$ and σ is the sampling mean and standard deviation of the data y respectively. Figure 8 showed the 99.7% of data are fallen within three standard deviation.

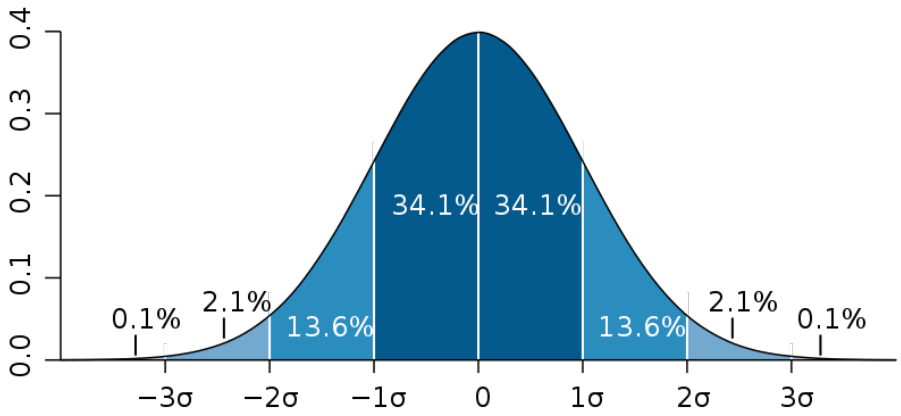


Figure 4: Standard deviation diagram.

4.2 Determination of model parameters

In this section, we proposed the models to estimate the maximum power generated $\hat{P}(t)$ at time index t, which consists of five models as summarized in table 1. The input variables of those models are consists of solar irradiance (I) and cell temperature (T).

Table 1: List of the proposed models

Model	Output	Input
Unbiased-polynomial model	$\hat{P}(t)$	$I(t), T(t)$
Biased-polynomial model	$\hat{P}(t)$	$I(t), T(t)$
ANN with Non-Lagging model	$\hat{P}(t)$	$I(t), T(t)$
ANN with Lagging model	$\hat{P}(t)$	$I(t), T(t), I(t-1), T(t-1)$
SVR model	$\hat{P}(t)$	$I(t), T(t)$

4.2.1 Polynomial models

The polynomial models used are expressed in (14) and (15) and thus can be rewritten as equation (21) and (24).

- **Unbiased-polynomial model**

$$\hat{P}_1(t) = x_1 I(t) + x_2 I(t)(T(t) - T_{ref}). \tag{21}$$

where x_1 and x_2 are constant parameters. The above model can be expressed to the linear models as

$$\hat{P}_1 = \begin{bmatrix} I & I(T - T_{\text{ref}}) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \triangleq X_1 \beta_1 \tag{22}$$

where $\hat{P}_1 \in \mathbb{R}^N$ is the output vector, $X_1 \in \mathbb{R}^{N \times 2}$ is the regressors matrix, and $\beta_1 \in \mathbb{R}^4$ is the regression vector. The model parameters x_1 and x_2 can be determined by solving linear least square problems:

$$\underset{x_1, x_2}{\text{minimize}} \frac{1}{N} \sum_{k=1}^{N} \Big(P(k) - \hat{P}_1(k) \Big)^2 \tag{23}$$

- **Biased-polynomial model**

$$\hat{P}_2(t) = x_1 + x_2 I(t) + x_3 T(t) + x_4 I(t)(T(t) - T_{ref}). \tag{24}$$

where x_1, x_2, x_3 and x_4 are constant parameters. The above model can be expressed to the linear models as

$$\hat{P}_2 = \begin{bmatrix} 1 & I & T - T_{\text{ref}} & I(T - T_{\text{ref}}) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \triangleq X_2 \beta_2 \tag{25}$$

where $\hat{P}_2 \in \mathbb{R}^N$ is the output vector, $X_2 \in \mathbb{R}^{N \times 4}$ is the regressors matrix, and $\beta_2 \in \mathbb{R}^4$ is the regression vector. The model parameters x_1, x_2, x_3 and x_4 can be determined by solving linear least square problems:

$$\underset{x_1, x_2, x_3, x_4}{\text{minimize}} \frac{1}{N} \sum_{k=1}^{N} \Big(P(k) - \hat{P}_2(k) \Big)^2 \tag{26}$$

where k is the index of data set and N is the number of data set.

4.2.2 Artificial Neural Networks

In order to create PV model for prediction with Artificial Neural Networks (ANN). We use Python as the main computer language with library such as Pandas for reorganized table, Matplotlib for Visualization, and Tensorflow for modeling Artificial Neural Networks (ANN). From the derivation of PV equivalent circuit model, we consider only irradiance (I) and Temperature (T) as the inputs (features) in "Input layer" and power (P) as the output in "Output layer". We use ADAM optimization as the optimizer and Rectified Linear Unit (ReLU) function as the activation function in both models.The ReLU function returns 0 if the input is the negative value and returns any positive value x back. So it can be written as

$$f(x) = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases} \tag{27}$$

In this section will propose the two ANN model that used the lagging data and non-lagging data as input.

- **ANN with Non-Lagging**

  This model is used only the input data at time index t for estimate the power generation $\hat{P}(t)$ at time index t.

$$\hat{P}_3(t) = f_3(T(t), I(t)). \tag{28}$$

where f_3 is the ANN model that has the structure followed in table 2.

Table 2: Model structure of ANN with non-lagging input

Layer index	Number of node	Activation function
1 (Input)	2	ReLU
2 (Hidden)	3	ReLU
3 (Output)	1	ReLU

The model parameters of ANN with non-lagging input is consists of 13 parameters.

$$\underset{W_i,b_i}{\text{minimize}} \frac{1}{N} \sum_{k=1}^{N} \Big(P(k) - \hat{P}_3(t)\Big)^2 \quad \text{for} \quad i = 1, 2. \tag{29}$$

- **ANN with Lagging**

  This model is used the input data at time index t and $t-1$ for estimate the power generation $\hat{P}(t)$ at time index t.

$$\hat{P}_4(t) = f_4(T(t), I(t), T(t-1), I(t-1)). \tag{30}$$

where f_3 is the ANN model that has the structure followed in table 3.

Table 3: Model structure of ANN with lagging input

Layer index	Number of node	Activation function
1 (Input)	4	ReLU
2 (Hidden)	3	ReLU
3 (Output)	1	ReLU

The model parameters of ANN with lagging input is consists of 19 parameters.

$$\underset{W_i,b_i}{\text{minimize}} \frac{1}{N} \sum_{k=1}^{N} \Big(P(k) - \hat{P}_4(t)\Big)^2 \quad \text{for} \quad i = 1, 2. \tag{31}$$

4.2.3 Support Vector Regression

This proposed model, we consider only the solar irradiance and cell temperature as the inputs (features) and power generated as the output at only time index t. The estimation equation is given by

$$\hat{P}_5(k) = \sum_{k=1}^{N} w_k \mathrm{K}(x_k, x) + b, \tag{32}$$

$$x_k = \begin{bmatrix} I(k) \\ T(k) \end{bmatrix}, \qquad x = \begin{bmatrix} I(1) & T(1) \\ I(2) & T(2) \\ \vdots & \vdots \\ I(k) & T(k) \end{bmatrix}$$

where the approximation function $\hat{P}_5(k)$ is represented as a sum of N radial basis functions, each associated with a different center x_k, weighted by an appropriate coefficient w_k, and K is the kernel function.

In order to create PV model for prediction with Support Vector Regression (SVR). We use Python as the main computer language with library such as Pandas for reorganized table, Matplotlib for Visualization and SKlearn for Support Vector Regression (SVR) Algorithm to minimize the optimization problem as follow:

$$\underset{w,b}{\text{minimize}} \frac{1}{2}\|w\|^2 \ + C\sum_{k=1}^{N}(\xi(k)+\xi^*(k)) \tag{33}$$

subject to

$$\begin{aligned} y(k) - \langle w, x(k)\rangle - b \leq \varepsilon + \xi(k), \qquad & \text{for} \quad k = 1, 2, ..., N, \\ \langle w, x(k)\rangle + b - y(k) \leq \varepsilon + \xi^*(k), \qquad & \text{for} \quad k = 1, 2, ..., N, \\ \xi(k), \xi^*(k) \geq 0, \qquad & \text{for} \quad k = 1, 2, ..., N. \end{aligned}$$

$$|\xi|_\varepsilon := \begin{cases} 0, & \text{if} |\xi| \leq \varepsilon \\ |\xi| - \varepsilon, & \text{Otherwise} \end{cases}$$

This model, we use Grid search Cross Validation from Sklearn library and obtain the best values of each parameters to fit the model.

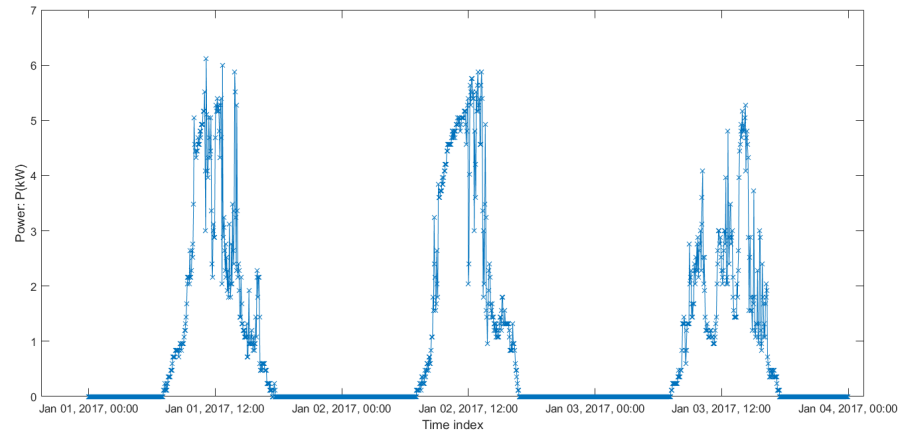
4.3 Model validation

To measure the performance of the models, we comparing the actual power (P) and estimated power ($\hat{P}$) of five models by using root mean-squared error(RMSE) as equation (18).

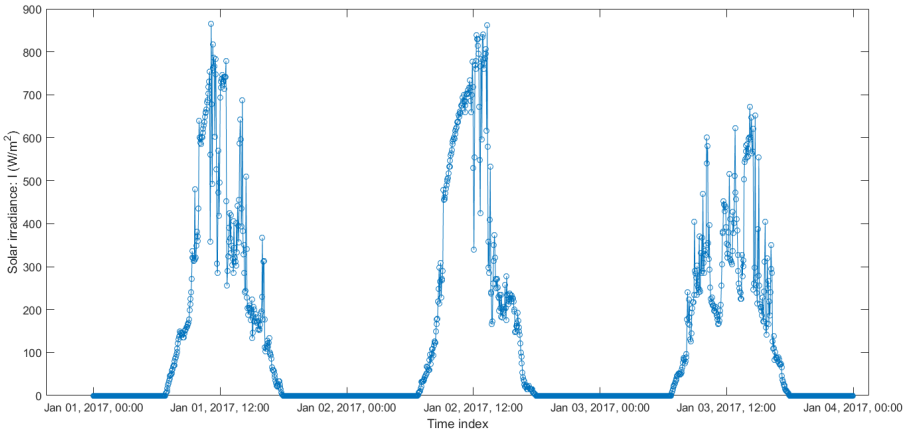
5 Experiments

5.1 Data description

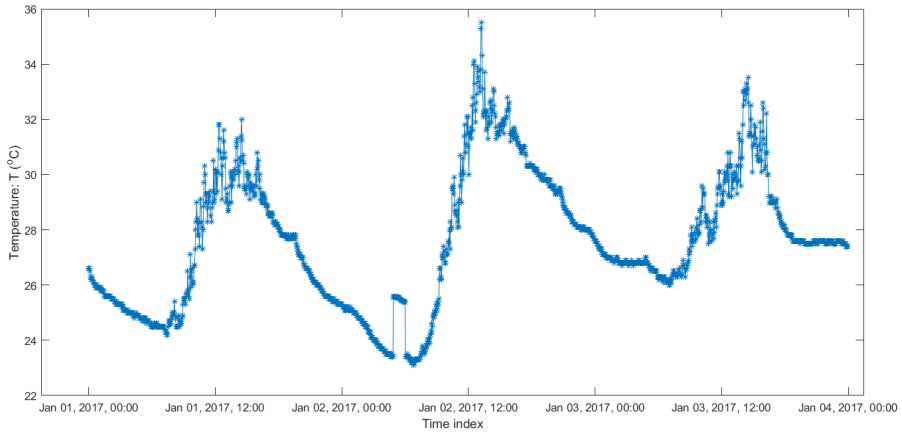
The 8kW solar power module data are collected from the solar site at EE building, Chulalongkorn university since Jan 1, 2017 until Jun 30, 2018 on 3 minutes sample periods.The data set is consist of solar generated power (P) in kW, solar irradiance (I) in W/m^2, and cell temperature (T) data in C°. An example of the time series plots of these data on January 1 to January 5, 2017 are showed in figure 5.



(a) Generated power data



(b) Solar irradiance data



(c) Cell temperature data

Figure 5: Time series plots from January 1 to January 5, 2017.

5.2 Outliers removing

An example of the time series plots of normalized data I and P are shown in figure 6. The plot on figure 6 shown the measurement data P and I which are normalized by $\bar{P}$ and $\bar{I}$ respectively. The figure 7 is shows the result from (20). The outliers are showed in both plots between time of 6:30 to 11:00 AM in Apr 17.

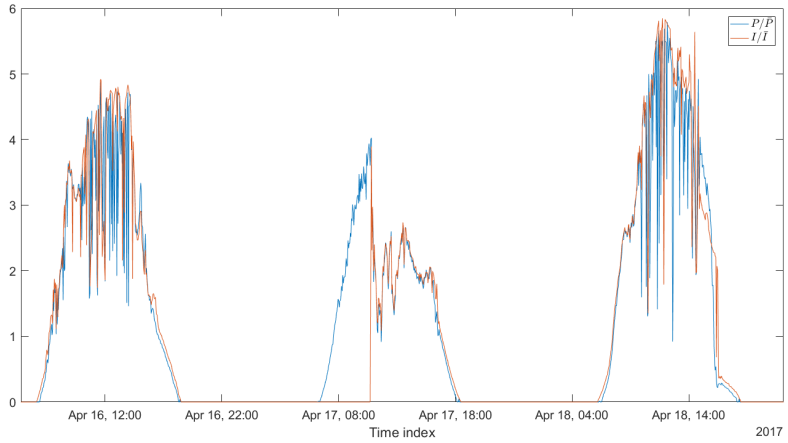


Figure 6: Time series plots of normalized irradiance data and power data.

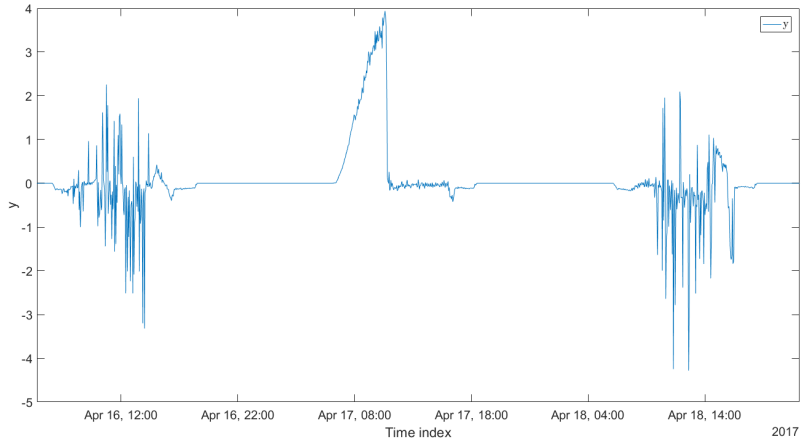


Figure 7: Time series plots of difference of normalized irradiance data and power data.

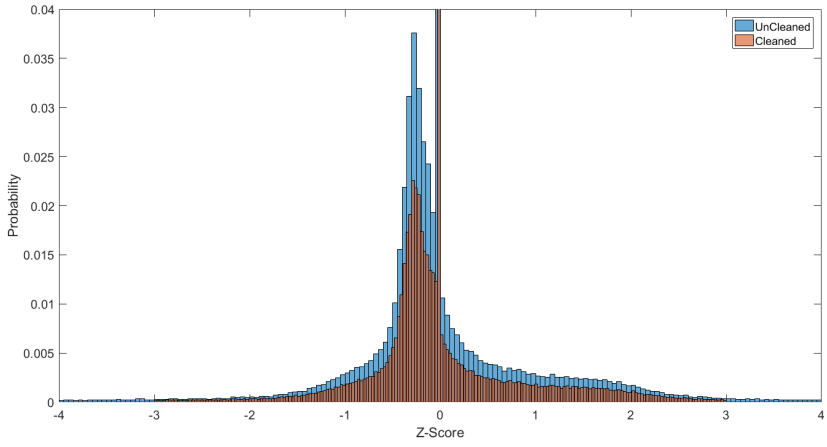


Figure 8: Distribution of uncleaned and cleaned data with Z-Score.

5.3 Experiment settings

In this experiment, we have studied on two kinds of data set which are data without outliers removed (uncleaned dataset) and with outliers removed (cleaned dataset). We choose to deleted outliers that have Z-score more than 3σ because, We need data samples as much as possible for training ANN. So that, about 99.73 percent of the data set will be kept in cleaned dataset. For uncleaned dataset, we split 80% of uncleaned dataset(208,228 samples) into training set and 20% of uncleaned dataset(51,751 samples) into validation set. For cleaned dataset, we split 80% of cleaned dataset(200,508 samples) into training set and 20% of cleaned dataset(49,610 samples) into validation set.

5.4 Experiment Results

The RMSE of estimation of five models are shown in table 4 and a example of time series plots of the power generated fits on clear sky day and cloudy day are showed in figure 9 and 10 respectively.

Table 4: Root mean squared error of the models

Model	Root Mean Squared Error (kW)			
	Uncleaned data		Cleaned data	
	Traning	Validation	Traning	Validation
Unbiased-polynomial model	0.3886	0.3732	0.2258	0.2249
Biased-polynomial model	0.3879	0.3728	0.2245	0.2234
Artificial Neural Networks with Non-Lagging	0.3521	0.3331	0.2399	0.2463
Artificial Neural Networks with Lagging	0.3497	0.3323	0.2138	0.2228
Support Vector Regression	0.3543	0.3351	0.2197	0.2328

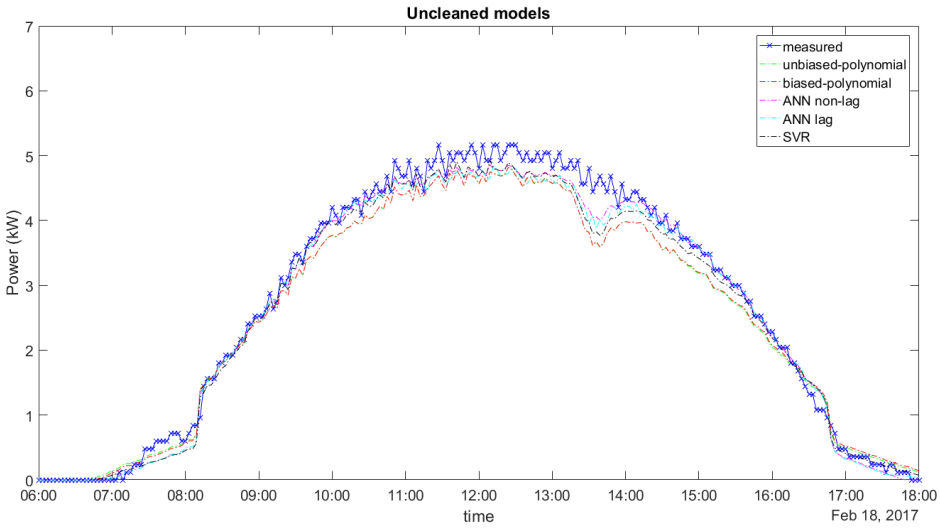


Figure 9: Time series plots of the power generated fits on clear sky day from uncleaned models.

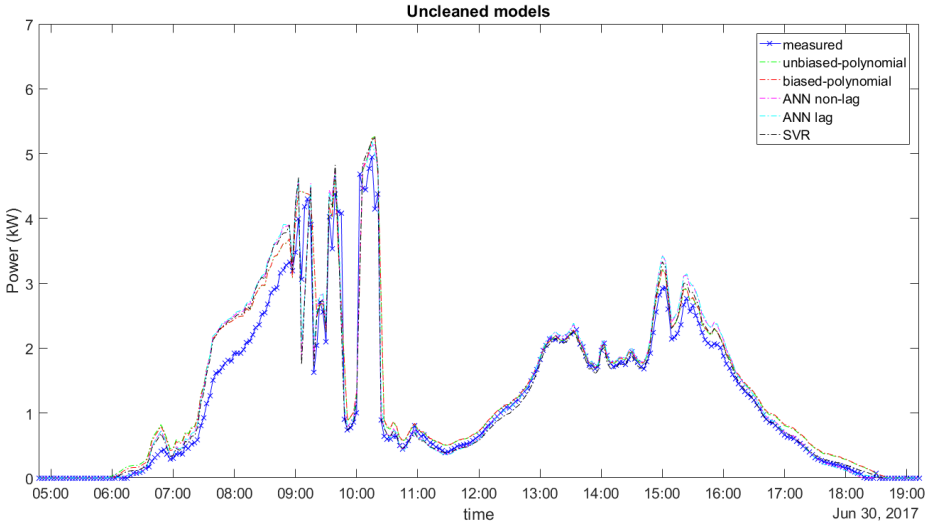


Figure 10: Time series plots of the power generated fits on cloudy day from uncleaned models.

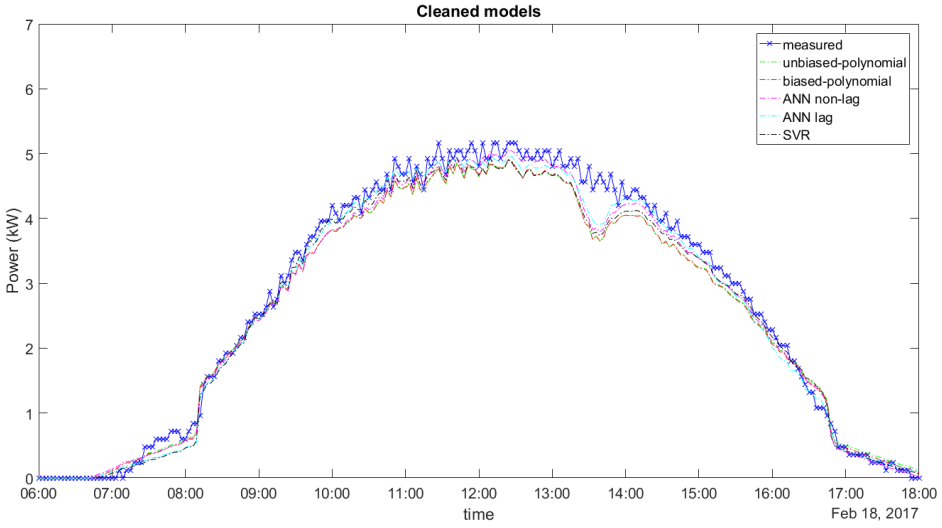


Figure 11: Time series plots of the power generated fits on clear sky day from cleaned models.

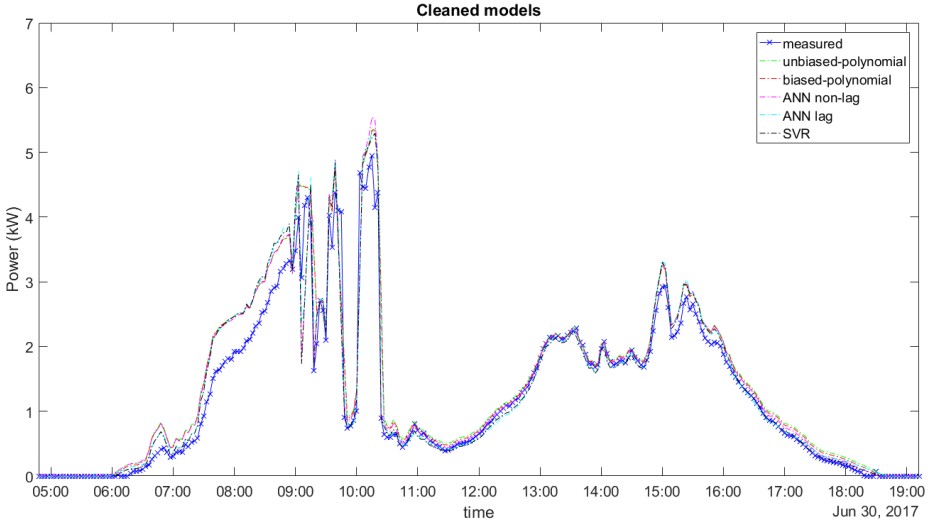


Figure 12: Time series plots of the power generated fits on cloudy day from cleaned models.

The results on table 4 shown the RMSE values from five models with uncleaned and cleaned data set. Both data sets have split the data into training and validation data set. The results on both ANN models and SVR model showed the values of RMSE are lower than the polynomial models. ANN with lagging input model gives the best result in both scenarios (cleaned and uncleaned data) which is directly affected from lagging inputs.

6 Conclusions

This project showed and compared the estimated results from five models to find the proper method for estimates the generated power from solar cell. The results showed the significant improvement when comparing with the ANN models, SVR model with the polynomial model. However, the cost for training the SVR model is greater than ANN models, but the performance is not significantly different. In addition, both ANN models also give the better performance over SVR model with lower runtime. This could be the effect of high computational cost in kernel of SVR model.

References

[1] A. Chouder, S. Silvestre, N. Sadaoui, and L. Rahmani. Modeling and simulation of a grid connected pv system based on the evaluation of main pv module parameters. *Simulation Modelling Practice and Theory*, 20(1):46–58, 2012.

[2] O. Gergaud, B. Multon, and H. Ahmed. Analysis and experimental validation of various photovoltaic system models. In *ELECTRIMACS*, page 6p, 2002.

[3] D. Rekioua and E. Matagne. Modeling of solar irradiance and cells. In *Optimization of Photovoltaic Power Systems*, pages 31–87. Springer, 2012.

[4] M. Samanta, B. Srikanth, and J. Yerrapragada. Short-term power forecasting of solar pv systems using machine learning techniques,(nd).

[5] A. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, 2004.

[6] M. Villalva, J. Gazoli, and E. Ruppert Filho. Comprehensive approach to modeling and simulation of photovoltaic arrays. *IEEE Transactions on power electronics*, 24(5):1198–1208, 2009.

Appendix A

Listing 1: DataPreparation.m

```
1
2  clear;
3  load('..\dataset\imputed_measurement_2017to2018.mat');
4
5  InvValidation = [361 8 97 296]; % Day index for validation dataset.
6
7  DataSetRaw = imputed_measurement_2017to2018(2);
8  DataSetRaw.OldIndex      = (1:length(DataSetRaw.TimeIndex))';
9  [ind_train, ind_test]    = splitdata(DataSetRaw, 1, 'slected days', InvValidation);
10
11 % Create validation dataset.
12 DataSetValidation        = MoveData(DataSetRaw, ind_test);
13 DataSetValidation        = CreateLaggingIndex(DataSetValidation,1);
14
15 DataSetUnCleaned         = MoveData(DataSetRaw, ind_train);
16
17 % Removeing the outliers data.
18 ind_cut                  = OutlierCut(DataSetUnCleaned);
19 DataSetCleaned           = MoveData(DataSetUnCleaned, ind_cut);
20
21 % Split dataset to 80-20, train-test.
22 [ind_train_UnCleaned , ind_test_UnCleaned]  = splitdata(DataSetUnCleaned, 0.8, 'rand
       over days');
23 [ind_train_Cleaned, ind_test_Cleaned]       = splitdata(DataSetCleaned, 0.8, 'rand
      over days');
24
25 % Move data to Train and Test.
26 DataSetUnCleaned_Train      = MoveData(DataSetUnCleaned, ind_train_UnCleaned);
27 DataSetUnCleaned_Test       = MoveData(DataSetUnCleaned, ind_test_UnCleaned);
28 DataSetCleaned_Train        = MoveData(DataSetCleaned, ind_train_Cleaned);
29 DataSetCleaned_Test         = MoveData(DataSetCleaned, ind_test_Cleaned);
30
31 % Create lagging index matrix.
32 DataSetUnCleaned_Train      = CreateLaggingIndex(DataSetUnCleaned_Train,1);
33 DataSetUnCleaned_Test       = CreateLaggingIndex(DataSetUnCleaned_Test,1);
34 DataSetCleaned_Train        = CreateLaggingIndex(DataSetCleaned_Train,1);
35 DataSetCleaned_Test         = CreateLaggingIndex(DataSetCleaned_Test,1);
36
37 % Calculate the data proportion after splits and cleaned.
38 Proportion_UnCleaned        = [length(DataSetUnCleaned_Train.LaggingIndex) length(
       DataSetUnCleaned_Test.LaggingIndex)]/(length(DataSetUnCleaned_Train.LaggingIndex)
       + length(DataSetUnCleaned_Test.LaggingIndex));
39 Proportion_Cleaned          = [length(DataSetCleaned_Train.LaggingIndex) length(
       DataSetCleaned_Test.LaggingIndex)]/(length(DataSetCleaned_Train.LaggingIndex) +
       length(DataSetCleaned_Test.LaggingIndex));
40 Proportion_UnCleanedOfRaw   = [length(DataSetUnCleaned_Train.LaggingIndex) length(
       DataSetUnCleaned_Test.LaggingIndex)]/length(DataSetRaw.TimeIndex);
41 Proportion_CleanedOfRaw     = [length(DataSetCleaned_Train.LaggingIndex) length(
       DataSetCleaned_Test.LaggingIndex)]/length(DataSetRaw.TimeIndex);
42
43 % Saving dataset into files.
44 save('../dataset/DataSetUnCleaned_Train.mat','DataSetUnCleaned_Train');
45 save('../dataset/DataSetUnCleaned_Test.mat','DataSetUnCleaned_Test');
46 save('../dataset/DataSetCleaned_Train.mat','DataSetCleaned_Train');
47 save('../dataset/DataSetCleaned_Test.mat','DataSetCleaned_Test');
48 save('../dataset/DataSetValidation.mat','DataSetValidation');
49 clear all;
```

Listing 2: PlotResults.m

```
1  clear all; close all;
```

```
2
3  load('..\dataset\DataSetValidation.mat');
4  load('..\results\validation\Y_HAT_POLY1_UNCLEANED.mat');
5  load('..\results\validation\Y_HAT_POLY2_UNCLEANED.mat');
6  load('..\results\validation\Y_HAT_ANN_NONLAG_UNCLEANED.mat');
7  load('..\results\validation\Y_HAT_ANN_LAG_UNCLEANED.mat');
8  load('..\results\validation\Y_HAT_SVR_UNCLEANED.mat');
9
10 load('..\results\validation\Y_HAT_POLY1_CLEANED.mat');
11 load('..\results\validation\Y_HAT_POLY2_CLEANED.mat');
12 load('..\results\validation\Y_HAT_ANN_NONLAG_CLEANED.mat');
13 load('..\results\validation\Y_HAT_ANN_LAG_CLEANED.mat');
14 load('..\results\validation\Y_HAT_SVR_CLEANED.mat');
15
16 y = DataSetValidation.P(DataSetValidation.LaggingIndex(:,1));
17
18 figure;
19 t = DataSetValidation.TimeIndex(DataSetValidation.LaggingIndex(:,1));
20
21 plot(t, y, 'xb-', t, Y_HAT_POLY1_UNCLEANED, 'g-.', t, Y_HAT_POLY2_UNCLEANED, 'r-.',
       t, Y_HAT_ANN_NONLAG_UNCLEANED, 'm-.', t, Y_HAT_ANN_LAG_UNCLEANED, 'c-.', t,
       Y_HAT_SVR_UNCLEANED, 'k-.');
22 ylabel('Power (kW)'); xlabel('time'); title('Uncleaned models');
23 legend('measured', 'unbiased-polynomial', 'biased-polynomial', 'ANN non-lag', 'ANN
       lag', 'SVR');
24 set(gca, 'fontsize', 16);
25 ylim([0 7]);
26 %xlim([736744.25 736744.75]);
27 %xlim([736876.20 736876.80]);
28 %xlim([737064.25 737064.75]);
29 xlim([737208.20 737208.80]);
30
31 figure;
32 plot(t, y, 'xb-', t, Y_HAT_POLY1_CLEANED, 'g-.', t, Y_HAT_POLY2_CLEANED, 'r-.', t,
       Y_HAT_ANN_NONLAG_CLEANED, 'm-.', t, Y_HAT_ANN_LAG_CLEANED, 'c-.', t,
       Y_HAT_SVR_CLEANED, 'k-.');
33 ylabel('Power (kW)'); xlabel('time'); title('Cleaned models');
34 legend('measured', 'unbiased-polynomial', 'biased-polynomial', 'ANN non-lag', 'ANN
       lag', 'SVR');
35 set(gca, 'fontsize', 16);
36 ylim([0 7]);
37 %xlim([736744.25 736744.75]);
38 %xlim([736876.20 736876.80]);
39 %xlim([737064.25 737064.75]);
40 xlim([737208.20 737208.80]);
```

Listing 3: splitdata.m

```
1  %% Split data sets
2
3  function[ind_train, ind_test] = splitdata(x, factor_train, METHOD, SlectedDays)
4
5  num_record = length(x.TimeIndex);
6
7  switch METHOD
8      case 'rand'
9          ind             = randperm(num_record); % permute indices (mix over days and
               times)
10         % we split training and validation with ratio of factor
11         ind_train       = ind(1:ceil(factor_train * num_record));
12         ind_test        = ind(ceil(factor_train * num_record)+1:end);
13
14     case 'rand over days'
15         listofdayyear   = unique([day(x.TimeIndex,'dayofyear') year(x.TimeIndex)], '
               rows');
```

```
        numday          = length(listofdayyear); % number of year in this data
        ind             = randperm(numday);
        day_train       = ind(1:ceil(factor_train*numday)); % day index for training
        ind_all         = (1:num_record);
        ind_train       = ind_all(ismember([day(x.TimeIndex, 'dayofyear') year(x.
            TimeIndex)], listofdayyear(day_train,:), 'rows'));
        ind_test        = setdiff(ind_all, ind_train);

    case 'slected days'
        listofdayyear   = unique([day(x.TimeIndex,'dayofyear') year(x.TimeIndex)], '
            rows');
        numday          = length(listofdayyear); % number of year in this data
        ind             = 1:numday;
        ind_all         = (1:num_record);
        ind_test        = ind_all(ismember([day(x.TimeIndex, 'dayofyear') year(x.
            TimeIndex)], listofdayyear(SlectedDays,:), 'rows'));
        ind_train       = setdiff(ind_all, ind_test);
end
end
```

Listing 4: MoveData.m

```
% Move dataset to new dataset with selected index.
function NewData = MoveData(OldData, index)
    % index : selected index.

    NewData.OldIndex = OldData.OldIndex(index);
    NewData.TimeIndex = OldData.TimeIndex(index);
    NewData.P = OldData.P(index);
    NewData.I = OldData.I(index);
    NewData.T = OldData.T(index);
    NewData.UV = OldData.UV(index);
    NewData.WS = OldData.WS(index);
    NewData.RH = OldData.RH(index);
end
```

Listing 5: OutlierCut.m

```
function [ind] = OutlierCut(x)

% Calculate the sampling mean of P and I.
mu_P    = mean(x.P);
mu_I    = mean(x.I);

y       = (x.P/mu_P) - (x.I/mu_I);
mu_y    = mean(y);
sd_y    = std(y);

% Used the Z-Score = 3 for removing outliers.
tmp     = abs(y-mu_y)/sd_y <= 3;
ind     = find(tmp);
end
```

Listing 6: CreateLaggingIndex.m

```
% Create index matrix for lagging input
function NewDataset = CreateLaggingIndex(OldDataset, n)

    % OldDataset : Old Dataset.
    % n          : number of lagging order.
    % NewDataset : New Dataset after create lagging index and remove
    % non-continuly index.

    NewDataset = OldDataset; % Move dataset to new dataset
    ind = 1:length(OldDataset.OldIndex);
```

```
    tmp = [lagmatrix(OldDataset.OldIndex,0:n) lagmatrix(ind',0:n)];
    NewDataset.LaggingIndexOld = tmp(sum(tmp(:,1:n) - tmp(:,2:n+1),2) == n,:);

    % LaggingIndex for indexing with new dataset
    % LaggingIndexOld for indexing with raw dataset
    NewDataset.LaggingIndex = NewDataset.LaggingIndexOld(:,n+2:end);
    NewDataset.LaggingIndexOld = NewDataset.LaggingIndexOld(:,1:n+1);
end
```

Listing 7: linls.m

```

%% Linear model estimation using LS

function[model] = linls(y, X)
% solving LS: minimize || y - X\beta ||
% where y is output, and X is the refressor matrix

if size(y,1) ~= size(X,1)
    error('X and y must have the same numbers of rows');
end

if det(X'*X) <= 1e-4
    error('the regressor matrix is nearly singular');
end

N           = length(y);
beta        = X\y;
model.beta  = beta;
model.y     = y; % store the original output
model.yhat  = X*beta;
model.resid = y - model.yhat;
model.rmse  = norm(model.resid)/sqrt(N);

end
```

Listing 8: linls_yhat.m

```
%% Evaluation of errors
function[modeloutput] = linls_yhat(model, y, X)
% compute prediction and error from a linear model with corresponding
% regressor X and output y

[N,n] = size(X);

if size(y,1) ~= size(X,1)
    error('X and y must have the same numbers of rows');
end

if n ~= length(model.beta)
    error('number of parameter in the model do not match with regressor matrix');
end

modeloutput.y = y;
modeloutput.yhat = X*model.beta;
modeloutput.resid = y - modeloutput.yhat;
modeloutput.rmse = norm(modeloutput.resid)/sqrt(N);
end
```

Appendix B

Listing 9: RunPolynomialModels.m

```
1
2  clear all; close all;
3
4  load('../dataset/DataSetCleaned_Test.mat');
5  load('../dataset/DataSetCleaned_Train.mat');
6  load('../dataset/DataSetUnCleaned_Test.mat');
7  load('../dataset/DataSetUnCleaned_Train.mat');
8  load('../dataset/DataSetValidation.mat');
9
10 % Renames the dataset.
11 data_train          = DataSetUnCleaned_Train;
12 data_test           = DataSetUnCleaned_Test;
13 data_train_clean    = DataSetCleaned_Train;
14 data_test_clean     = DataSetCleaned_Test;
15 data_validation     = DataSetValidation;
16
17 Tref    = 25;   % Reference temperature.
18 N       = 1;  % Number of training.
19
20 Beta_M1 = [];
21 Beta_M2 = [];
22 Beta_M1_clean = [];
23 Beta_M2_clean = [];
24
25 RMSE_M1 = [];
26 RMSE_M2 = [];
27 RMSE_M1_clean = [];
28 RMSE_M2_clean = [];
29
30 numdata.train           = length(data_train.P);
31 numdata.test            = length(data_test.P);
32 numdata.alldata         = numdata.train + numdata.test;
33
34 numdata.train_clean     = length(data_train_clean.P);
35 numdata.test_clean      = length(data_test_clean.P);
36 numdata.alldata_clean   = numdata.train_clean + numdata.test_clean;
37
38 % Regressors of model 1 (Unbiased-polynomial model): X is for training, Z is for
       validation
39 X1 = [data_train.I/1000 (data_train.I/1000).*(data_train.T - Tref)/10];
40 X1_clean = [data_train_clean.I/1000 (data_train_clean.I/1000).*(data_train_clean.T-
       Tref)/10];
41
42 % Regressors of model 2 (Biased-polynomial model): X is for training, Z is for
       validation
43 X2 = [ones(numdata.train, 1) data_train.I/1000 (data_train.T-Tref)/10 (data_train.I
       /1000).*(data_train.T-Tref)/10];
44 X2_clean = [ones(numdata.train_clean, 1) data_train_clean.I/1000 (data_train_clean.T
       -Tref)/10 (data_train_clean.I/1000).*(data_train_clean.T-Tref)/10];
45
46 for k = 1:N
47     %% Prepare data for estimation and validation
48     ind                     = randperm(numdata.train);
49     ind_clean               = randperm(numdata.train_clean);
50
51
52     %% Model estimation
53     % Model 1
54     model1          = linls(data_train.P(ind), X1(ind,:));
55     model1_clean    = linls(data_train_clean.P(ind_clean), X1_clean(ind_clean,:));
56
57     % Model 2
58     model2          = linls(data_train.P(ind), X2(ind,:));
59     model2_clean    = linls(data_train_clean.P(ind_clean), X2_clean(ind_clean,:));
60
```

```
61      % Recording the results.
62      Beta_M1 = [Beta_M1 model1.beta];
63      Beta_M2 = [Beta_M2 model2.beta];
64
65      Beta_M1_clean = [Beta_M1_clean model1_clean.beta];
66      Beta_M2_clean = [Beta_M2_clean model2_clean.beta];
67
68      RMSE_M1          = [RMSE_M1; model1.rmse];
69      RMSE_M2          = [RMSE_M2; model2.rmse];
70      RMSE_M1_clean    = [RMSE_M1_clean; model1_clean.rmse];
71      RMSE_M2_clean    = [RMSE_M2_clean; model2_clean.rmse];
72
73  end
74
75  % Evaluate the averaged model parameters.
76  Beta_M1_Averaged          = mean(Beta_M1,2);
77  Beta_M2_Averaged          = mean(Beta_M2,2);
78  Beta_M1_clean_Averaged    = mean(Beta_M1_clean,2);
79  Beta_M2_clean_Averaged    = mean(Beta_M2_clean,2);
80
81  RMSE_M1_Averaged          = mean(RMSE_M1);
82  RMSE_M2_Averaged          = mean(RMSE_M2);
83  RMSE_M1_clean_Averaged    = mean(RMSE_M1_clean);
84  RMSE_M2_clean_Averaged    = mean(RMSE_M2_clean);
85
86  %Test
87  % Regressors of model 1: X is for training, Z is for validation
88  Z1 = [data_test.I/1000 (data_test.I/1000).*(data_test.T-Tref)/10];
89  Z1_clean = [data_test_clean.I/1000 (data_test_clean.I/1000).*(data_test_clean.T-Tref
        )/10];
90
91  % Regressors of model 2: X is for training, Z is for validation
92  Z2 = [ones(numdata.test, 1) data_test.I/1000 (data_test.T-Tref)/10 (data_test.I
        /1000).*(data_test.T-Tref)/10];
93  Z2_clean = [ones(numdata.test_clean, 1) data_test_clean.I/1000 (data_test_clean.T-
        Tref)/10 (data_test_clean.I/1000).*(data_test_clean.T-Tref)/10];
94
95  % Using the averaged model parameters for verify the models on test dataset.
96  model1.beta             = Beta_M1_Averaged;
97  model2.beta             = Beta_M2_Averaged;
98  model1_clean.beta       = Beta_M1_clean_Averaged;
99  model2_clean.beta       = Beta_M2_clean_Averaged;
100
101 output_model1           = linls_yhat(model1, data_test.P, Z1);
102 output_model1_clean     = linls_yhat(model1_clean, data_test_clean.P, Z1_clean);
103 output_model2           = linls_yhat(model2, data_test.P, Z2);
104 output_model2_clean     = linls_yhat(model2_clean, data_test_clean.P, Z2_clean);
105
106 % Printing the fitting results.
107 [RMSE_M1_Averaged RMSE_M2_Averaged; RMSE_M1_clean_Averaged RMSE_M2_clean_Averaged]
108 [output_model1.rmse output_model2.rmse; output_model1_clean.rmse output_model2_clean
        .rmse]
109
110 % Saving the results into files.
111 Y_HAT_POLY1_UNCLEANED = output_model1.yhat;
112 Y_HAT_POLY2_UNCLEANED = output_model2.yhat;
113 Y_HAT_POLY1_CLEANED   = output_model1_clean.yhat;
114 Y_HAT_POLY2_CLEANED   = output_model2_clean.yhat;
115
116 save('..\results\uncleaned\Y_HAT_POLY1_UNCLEANED.mat','Y_HAT_POLY1_UNCLEANED');
117 save('..\results\uncleaned\Y_HAT_POLY2_UNCLEANED.mat','Y_HAT_POLY2_UNCLEANED');
118 save('..\results\cleaned\Y_HAT_POLY1_CLEANED.mat','Y_HAT_POLY1_CLEANED');
119 save('..\results\cleaned\Y_HAT_POLY2_CLEANED.mat','Y_HAT_POLY2_CLEANED');
120
121 % Validation
```

```
% Regressors of model 1: X is for training, Z is for validation
ind = data_validation.LaggingIndex(:,1);
Z1 = [data_validation.I/1000 (data_validation.I/1000).*(data_validation.T-Tref)/10];

% Regressors of model 2: X is for training, Z is for validation
Z2 = [ones(length(data_validation.P), 1) data_validation.I/1000 (data_validation.T-
    Tref)/10 (data_validation.I/1000).*(data_validation.T-Tref)/10];

% Using the averaged model parameters for verify the models on test dataset.
output_model1           = linls_yhat(model1, data_validation.P(ind), Z1(ind,:));
output_model1_clean     = linls_yhat(model1_clean, data_validation.P(ind), Z1(ind,:)
    );
output_model2           = linls_yhat(model2, data_validation.P(ind), Z2(ind,:));
output_model2_clean     = linls_yhat(model2_clean, data_validation.P(ind), Z2(ind,:)
    );

% Saving the results into files.
Y_HAT_POLY1_UNCLEANED = output_model1.yhat;
Y_HAT_POLY2_UNCLEANED = output_model2.yhat;
Y_HAT_POLY1_CLEANED   = output_model1_clean.yhat;
Y_HAT_POLY2_CLEANED   = output_model2_clean.yhat;

save('..\results\validation\Y_HAT_POLY1_UNCLEANED.mat','Y_HAT_POLY1_UNCLEANED');
save('..\results\validation\Y_HAT_POLY2_UNCLEANED.mat','Y_HAT_POLY2_UNCLEANED');
save('..\results\validation\Y_HAT_POLY1_CLEANED.mat','Y_HAT_POLY1_CLEANED');
save('..\results\validation\Y_HAT_POLY2_CLEANED.mat','Y_HAT_POLY2_CLEANED');

% clear all;
```

Listing 10: ANN(T,T)-Datasets.py

```
## 2102531_System Identification Term Project

### Artificial Neural Networks(ANN)
    ##### INPUT:  Irradiance(t) and Temperature(t)
    ##### OUTPUT: Power(t)

#1. Import Library

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.io
import tensorflow as tf
import keras


#2. Data Preparation

feature=pd.read_csv("feature.csv",header=None)
feature.columns=[["Power","Irradiance","Temp","UV","WS","RH"]]
time=pd.read_csv("time.csv",header=None)


time_list=[]
for n in range(len(time)):
    time_sample=str(time.iloc[n,:][0])
    for i in range(len(time.iloc[n,:])-1):
        time_sample=time_sample+str(time.iloc[n,:][i+1])
    time_list.append(time_sample)

seperate_list=[]
for j in range(len(time_list)):
    seperate=time_list[j].split()
    seperate_list.append(seperate)
```

```
35
36 time_df=pd.DataFrame(seperate_list)
37 time_df.columns=[["Date","Time"]]
38 time_df.to_pickle("time_df.pickle")
39 time_df.head()
40
41 data=pd.concat([time_df,feature],axis=1)
42 data.head()
43
44
45 #3. Data Pre-Processing
46
47 traindata = scipy.io.loadmat('DataSetCleaned_Train.mat')
48 testdata = scipy.io.loadmat('DataSetCleaned_Test.mat')
49 valdata = scipy.io.loadmat('DataSetValidation.mat')
50
51 nonlag_trainindex=[]
52 lag_trainindex=[]
53 nonlag_testindex=[]
54 lag_testindex=[]
55 nonlag_valindex=[]
56 lag_valindex=[]
57 for i in range(len(traindata['DataSetCleaned_Train']['LaggingIndexOld'][0][0])):
58     nonlag_trainindex.append(traindata['DataSetCleaned_Train']['LaggingIndexOld'
        ][0][0][i][0])
59     lag_trainindex.append(traindata['DataSetCleaned_Train']['LaggingIndexOld'
        ][0][0][i][1])
60
61 for j in range(len(testdata['DataSetCleaned_Test']['LaggingIndexOld'][0][0])):
62     nonlag_testindex.append(testdata['DataSetCleaned_Test']['LaggingIndexOld'
        ][0][0][j][0])
63     lag_testindex.append(testdata['DataSetCleaned_Test']['LaggingIndexOld'][0][0][j
        ][1])
64
65 for k in range(len(valdata['DataSetValidation']['LaggingIndexOld'][0][0])):
66     nonlag_valindex.append(valdata['DataSetValidation']['LaggingIndexOld'][0][0][k
        ][0])
67     lag_valindex.append(valdata['DataSetValidation']['LaggingIndexOld'][0][0][k][1])
68
69
70 #Minus Index for Python
71 lag_trainindex=np.array(lag_trainindex).astype('int64')-np.array(1)
72 nonlag_trainindex=np.array(nonlag_trainindex).astype('int64')-np.array(1)
73 lag_testindex=np.array(lag_testindex).astype('int64')-np.array(1)
74 nonlag_testindex=np.array(nonlag_testindex).astype('int64')-np.array(1)
75 lag_valindex=np.array(lag_valindex).astype('int64')-np.array(1)
76 nonlag_valindex=np.array(nonlag_valindex).astype('int64')-np.array(1)
77
78 # For Training Set
79 training_set=data[data.index.isin(nonlag_trainindex)].reset_index(drop=True)
80 training_set.head()
81
82 # For Test Set
83 test_set=data[data.index.isin(nonlag_testindex)].reset_index(drop=True)
84 test_set.head()
85
86 # For Validation Set
87 val_set=data[data.index.isin(nonlag_valindex)].reset_index(drop=True)
88 val_set.head()
89
90 # Normalization
91 training_set.iloc[:,3:4]=training_set.iloc[:,3:4]/1000
92 training_set.iloc[:,4:5]=training_set.iloc[:,4:5]/10
93 test_set.iloc[:,3:4]=test_set.iloc[:,3:4]/1000
94 test_set.iloc[:,4:5]=test_set.iloc[:,4:5]/10
```

```
val_set.iloc[:,3:4]=val_set.iloc[:,3:4]/1000
val_set.iloc[:,4:5]=val_set.iloc[:,4:5]/10


x_train=training_set.iloc[:,3:5]
y_train=training_set.iloc[:,2:3]
x_test=test_set.iloc[:,3:5]
y_test=test_set.iloc[:,2:3]
x_val=val_set.iloc[:,3:5]
y_val=val_set.iloc[:,2:3]


#4. Artificial Neural Networks with Non-Lagging Model

model= keras.Sequential([
                        keras.layers.Dense(3,activation=tf.nn.relu,input_shape=(2,),
                                           kernel_initializer=keras.initializers.
                                               glorot_uniform(seed=2)),
                        keras.layers.Dense(1)
])
optimizer=tf.train.AdamOptimizer(learning_rate=0.001)
model.compile(loss='mse',optimizer=optimizer,metrics=['mse'])
model.summary()


early_stop = keras.callbacks.EarlyStopping(monitor='mean_squared_error', patience=5)
history = model.fit(x_train,y_train,epochs=100,callbacks=[early_stop],
    validation_data=(x_test,y_test))


#plot MSE
plt.figure()
plt.xlabel('Epoch')
plt.ylabel('Mean Squared Error')
plt.plot(history.epoch, np.array(history.history['mean_squared_error']),
label='Train Loss')
plt.plot(history.epoch, np.array(history.history['val_mean_squared_error']),
label='Validation Loss')
plt.legend()


train_predictions = model.predict(x_train).flatten()
test_predictions = model.predict(x_test).flatten()

#plot True Values & predictions
x=[x for x in range(480*1)]
plt.figure(figsize=(20,10))
plt.plot(x,np.array(y_test)[0:480*1,],label="True Values",marker='o')
plt.plot(x,test_predictions[0:480*1,],label="Prediction",marker='o')
plt.legend(['True Value','Prediction'])
plt.ylabel('Power')
plt.xlabel('TimeSample')
plt.title("Artificial Neural Networks with Non-lagging in a day")



mse_train=sum((np.array(y_train.Power).squeeze()-train_predictions)**2)/len(y_train)
mse_test=sum((np.array(y_test.Power).squeeze()-test_predictions)**2)/len(y_test)
print("MSE of Training set:"+"\t"+str(mse_train))
print("MSE of Test set:"+"\t"+str(mse_test))
```

Listing 11: ANN(T,T)-Datasets-NOTCLEAN.py

```
## 2102531_System Identification Term Project

```

```
### Artificial Neural Networks(ANN)
    ##### INPUT:  Irradiance(t) and Temperature(t)
    ##### OUTPUT: Power(t)

#1. Import Library

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.io
import tensorflow as tf
import keras


#2. Data Preparation

feature=pd.read_csv("feature.csv",header=None)
feature.columns=[["Power","Irradiance","Temp","UV","WS","RH"]]
time=pd.read_csv("time.csv",header=None)


time_list=[]
for n in range(len(time)):
    time_sample=str(time.iloc[n,:][0])
    for i in range(len(time.iloc[n,:])-1):
        time_sample=time_sample+str(time.iloc[n,:][i+1])
    time_list.append(time_sample)

seperate_list=[]
for j in range(len(time_list)):
    seperate=time_list[j].split()
    seperate_list.append(seperate)

time_df=pd.DataFrame(seperate_list)
time_df.columns=[["Date","Time"]]
time_df.to_pickle("time_df.pickle")
time_df.head()

data=pd.concat([time_df,feature],axis=1)
data.head()


#3. Data Pre-Processing

traindata = scipy.io.loadmat('DataSetUnCleaned_Train.mat')
testdata = scipy.io.loadmat('DataSetUnCleaned_Test.mat')
valdata = scipy.io.loadmat('DataSetValidation.mat')

nonlag_trainindex=[]
lag_trainindex=[]
nonlag_testindex=[]
lag_testindex=[]
nonlag_valindex=[]
lag_valindex=[]
for i in range(len(traindata['DataSetUnCleaned_Train']['LaggingIndexOld'][0][0])):
    nonlag_trainindex.append(traindata['DataSetUnCleaned_Train']['LaggingIndexOld'
        ][0][0][i][0])
    lag_trainindex.append(traindata['DataSetUnCleaned_Train']['LaggingIndexOld'
        ][0][0][i][1])

for j in range(len(testdata['DataSetUnCleaned_Test']['LaggingIndexOld'][0][0])):
    nonlag_testindex.append(testdata['DataSetUnCleaned_Test']['LaggingIndexOld'
        ][0][0][j][0])
    lag_testindex.append(testdata['DataSetUnCleaned_Test']['LaggingIndexOld'][0][0][
        j][1])
```

```
64
65 for k in range(len(valdata['DataSetValidation']['LaggingIndexOld'][0][0])):
66     nonlag_valindex.append(valdata['DataSetValidation']['LaggingIndexOld'][0][0][k
       ][0])
67     lag_valindex.append(valdata['DataSetValidation']['LaggingIndexOld'][0][0][k][1])
68
69 # Minus Index for Python
70 lag_trainindex=np.array(lag_trainindex).astype('int64')-np.array(1)
71 nonlag_trainindex=np.array(nonlag_trainindex).astype('int64')-np.array(1)
72 lag_testindex=np.array(lag_testindex).astype('int64')-np.array(1)
73 nonlag_testindex=np.array(nonlag_testindex).astype('int64')-np.array(1)
74 lag_valindex=np.array(lag_valindex).astype('int64')-np.array(1)
75 nonlag_valindex=np.array(nonlag_valindex).astype('int64')-np.array(1)
76
77
78 # For Training Set
79 training_set=data[data.index.isin(nonlag_trainindex)].reset_index(drop=True)
80 training_set.head()
81
82 #For Test Set
83 test_set=data[data.index.isin(nonlag_testindex)].reset_index(drop=True)
84 test_set.head()
85
86 # For Validation Set
87 val_set=data[data.index.isin(nonlag_valindex)].reset_index(drop=True)
88 val_set.head()
89
90 # Normalization
91 training_set.iloc[:,3:4]=training_set.iloc[:,3:4]/1000
92 training_set.iloc[:,4:5]=training_set.iloc[:,4:5]/10
93 test_set.iloc[:,3:4]=test_set.iloc[:,3:4]/1000
94 test_set.iloc[:,4:5]=test_set.iloc[:,4:5]/10
95 val_set.iloc[:,3:4]=val_set.iloc[:,3:4]/1000
96 val_set.iloc[:,4:5]=val_set.iloc[:,4:5]/10
97
98 x_train=training_set.iloc[:,3:5]
99 y_train=training_set.iloc[:,2:3]
100 x_test=test_set.iloc[:,3:5]
101 y_test=test_set.iloc[:,2:3]
102 x_val=val_set.iloc[:,3:5]
103 y_val=val_set.iloc[:,2:3]
104
105
106 #4. Artificial Neural Networks with Non-Lagging Model
107
108 model= keras.Sequential([
109                          keras.layers.Dense(3,activation=tf.nn.relu,input_shape=(2,),
110                                             kernel_initializer=keras.initializers.
                                                   glorot_uniform(seed=2)),
111                          keras.layers.Dense(1)
112 ])
113 optimizer=tf.train.AdamOptimizer(learning_rate=0.0001)
114 model.compile(loss='mse',optimizer=optimizer,metrics=['mse'])
115 model.summary()
116
117
118 early_stop = keras.callbacks.EarlyStopping(monitor='mean_squared_error', patience=5)
119 history = model.fit(x_train,y_train,epochs=100,callbacks=[early_stop],
        validation_data=(x_test,y_test))
120
121
122 #plot MSE
123 plt.figure()
124 plt.xlabel('Epoch')
125 plt.ylabel('Mean Squared Error')
```

```
plt.plot(history.epoch, np.array(history.history['mean_squared_error']),
label='Train Loss')
plt.plot(history.epoch, np.array(history.history['val_mean_squared_error']),
label='Validation Loss')
plt.legend()


train_predictions = model.predict(x_train).flatten()
test_predictions = model.predict(x_test).flatten()

#plot True Values & predictions
x=[x for x in range(480*1)]
plt.figure(figsize=(20,10))
plt.plot(x,np.array(y_test)[0:480*1,],label="True Values",marker='o')
plt.plot(x,test_predictions[0:480*1,],label="Prediction",marker='o')
plt.legend(['True Value','Prediction'])
plt.ylabel('Power')
plt.xlabel('TimeSample')
plt.title("Artificial Neural Networks with Non-lagging in a day")



mse_train=sum((np.array(y_train.Power).squeeze()-train_predictions)**2)/len(y_train)
mse_test=sum((np.array(y_test.Power).squeeze()-test_predictions)**2)/len(y_test)
print("MSE of Training set:"+"\t"+str(mse_train))
print("MSE of Test set:"+"\t"+str(mse_test))
```

Listing 12: ANN(T,T-1)-Datasets.py

```
# # 2102531_System Identification Term Project

### Artificial Neural Networks(ANN)
    ##### INPUT: Irradiance(t) and Temperature(t),Irradiance(t-1) and Temperature(t
        -1)
    ##### OUTPUT: Power(t)

#1. Import Library

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.io
import tensorflow as tf
import keras


#2. Data Preparation

feature=pd.read_csv("feature.csv",header=None)
feature.columns=[["Power","Irradiance","Temp","UV","WS","RH"]]
time=pd.read_csv("time.csv",header=None)


time_list=[]
for n in range(len(time)):
    time_sample=str(time.iloc[n,:][0])
    for i in range(len(time.iloc[n,:])-1):
        time_sample=time_sample+str(time.iloc[n,:][i+1])
    time_list.append(time_sample)

seperate_list=[]
for j in range(len(time_list)):
    seperate=time_list[j].split()
    seperate_list.append(seperate)

```

```
time_df=pd.DataFrame(seperate_list)
time_df.columns=[["Date","Time"]]
time_df.to_pickle("time_df.pickle")
time_df.head()

data=pd.concat([time_df,feature],axis=1)
data.head()


#3. Data Pre-Processing

traindata = scipy.io.loadmat('DataSetCleaned_Train.mat')
testdata = scipy.io.loadmat('DataSetCleaned_Test.mat')
valdata = scipy.io.loadmat('DataSetValidation.mat')


nonlag_trainindex=[]
lag_trainindex=[]
nonlag_testindex=[]
lag_testindex=[]
nonlag_valindex=[]
lag_valindex=[]
for i in range(len(traindata['DataSetCleaned_Train']['LaggingIndexOld'][0][0])):
    nonlag_trainindex.append(traindata['DataSetCleaned_Train']['LaggingIndexOld'
        ][0][0][i][0])
    lag_trainindex.append(traindata['DataSetCleaned_Train']['LaggingIndexOld'
        ][0][0][i][1])

for j in range(len(testdata['DataSetCleaned_Test']['LaggingIndexOld'][0][0])):
    nonlag_testindex.append(testdata['DataSetCleaned_Test']['LaggingIndexOld'
        ][0][0][j][0])
    lag_testindex.append(testdata['DataSetCleaned_Test']['LaggingIndexOld'][0][0][j
        ][1])

for k in range(len(valdata['DataSetValidation']['LaggingIndexOld'][0][0])):
    nonlag_valindex.append(valdata['DataSetValidation']['LaggingIndexOld'][0][0][k
        ][0])
    lag_valindex.append(valdata['DataSetValidation']['LaggingIndexOld'][0][0][k][1])


# Minus Index for Python
lag_trainindex=np.array(lag_trainindex).astype('int64')-np.array(1)
nonlag_trainindex=np.array(nonlag_trainindex).astype('int64')-np.array(1)
lag_testindex=np.array(lag_testindex).astype('int64')-np.array(1)
nonlag_testindex=np.array(nonlag_testindex).astype('int64')-np.array(1)
lag_valindex=np.array(lag_valindex).astype('int64')-np.array(1)
nonlag_valindex=np.array(nonlag_valindex).astype('int64')-np.array(1)

# For Training Set
training_set=data[data.index.isin(nonlag_trainindex)].reset_index(drop=True)
training_set.head()
temp_trainlag=data[data.index.isin(lag_trainindex)].Temp.reset_index(drop=True)
irradiance_trainlag=data[data.index.isin(lag_trainindex)].Irradiance.reset_index(
    drop=True)
col_names=['Date','Time','Power','Irradiance','Temp','UV','WS','RH','Irradiance_lag'
    ,'Temp_lag']
training_set=pd.concat([training_set,irradiance_trainlag,temp_trainlag],axis=1)
training_set.columns=col_names
training_set=training_set[['Date','Time','Power','Irradiance','Irradiance_lag','Temp
    ','Temp_lag','UV','WS','RH']]
training_set.head()


#For Test Set
test_set=data[data.index.isin(nonlag_testindex)].reset_index(drop=True)
```

```
test_set.head()
temp_testlag=data[data.index.isin(lag_testindex)].Temp.reset_index(drop=True)
irradiance_testlag=data[data.index.isin(lag_testindex)].Irradiance.reset_index(drop=True)
col_names=['Date','Time','Power','Irradiance','Temp','UV','WS','RH','Irradiance_lag','Temp_lag']
test_set=pd.concat([test_set,irradiance_testlag,temp_testlag],axis=1)
test_set.columns=col_names
test_set=test_set[['Date','Time','Power','Irradiance','Irradiance_lag','Temp','Temp_lag','UV','WS','RH']]
test_set.head()

# For Validation Set
val_set=data[data.index.isin(nonlag_valindex)].reset_index(drop=True)
val_set.head()
temp_vallag=data[data.index.isin(lag_valindex)].Temp.reset_index(drop=True)
irradiance_vallag=data[data.index.isin(lag_valindex)].Irradiance.reset_index(drop=True)
col_names=['Date','Time','Power','Irradiance','Temp','UV','WS','RH','Irradiance_lag','Temp_lag']
val_set=pd.concat([val_set,irradiance_vallag,temp_vallag],axis=1)
val_set.columns=col_names
val_set=val_set[['Date','Time','Power','Irradiance','Irradiance_lag','Temp','Temp_lag','UV','WS','RH']]
val_set.head()

# Normalization
training_set.iloc[:,3:5]=training_set.iloc[:,3:5]/1000
training_set.iloc[:,5:7]=training_set.iloc[:,5:7]/10
test_set.iloc[:,3:5]=test_set.iloc[:,3:5]/1000
test_set.iloc[:,5:7]=test_set.iloc[:,5:7]/10
val_set.iloc[:,3:5]=val_set.iloc[:,3:5]/1000
val_set.iloc[:,5:7]=val_set.iloc[:,5:7]/10


x_train=training_set.iloc[:,3:7]
y_train=training_set.iloc[:,2:3]
x_test=test_set.iloc[:,3:7]
y_test=test_set.iloc[:,2:3]
x_val=val_set.iloc[:,3:7]
y_val=val_set.iloc[:,2:3]


#4. Artificial Neural Networks with Lagging Model


model= keras.Sequential([
                         keras.layers.Dense(3,activation=tf.nn.relu,input_shape=(4,),
                             kernel_initializer=keras.initializers.glorot_uniform(seed
                             =2)),
                         keras.layers.Dense(1)
])


optimizer=tf.train.AdamOptimizer(learning_rate=0.001)
model.compile(loss='mse',optimizer=optimizer,metrics=['mse'])
model.summary()

early_stop = keras.callbacks.EarlyStopping(monitor='mean_squared_error', patience=5)
early_stop
history = model.fit(x_train,y_train, epochs=100,callbacks=[early_stop],
    validation_data=(x_test,y_test))


#plot MSE
```

```
plt.figure()
plt.xlabel('Epoch')
plt.ylabel('Mean Squared Error')
plt.plot(history.epoch, np.array(history.history['mean_squared_error']),
label='Train Loss')
plt.plot(history.epoch, np.array(history.history['val_mean_squared_error']),
label='Validation Loss')
plt.legend()


#plot True Values & predictions

train_predictions = model.predict(x_train).flatten()
test_predictions = model.predict(x_test).flatten()

#plot True Values & predictions
x=[x for x in range(480*1)]
plt.figure(figsize=(20,10))
plt.plot(x,np.array(y_test)[0:480*1,],label="True Values",marker='o')
plt.plot(x,test_predictions[0:480*1,],label="Prediction",marker='o')
plt.legend(['True Value','Prediction'])
plt.ylabel('Power')
plt.xlabel('TimeSample')
plt.title("Artificial Neural Networks with Non-lagging in a day")


mse_train=sum((np.array(y_train.Power).squeeze()-train_predictions)**2)/len(y_train)
mse_test=sum((np.array(y_test.Power).squeeze()-test_predictions)**2)/len(y_test)
print("MSE of Training set:"+"\t"+str(mse_train))
print("MSE of Test set:"+"\t"+str(mse_test))
```

Listing 13: ANN(T,T-1)-Datasets-NOTCLEAN.py

```
# # 2102531_System Identification Term Project

### Artificial Neural Networks(ANN)
    ##### INPUT: Irradiance(t) and Temperature(t),Irradiance(t-1) and Temperature(t
        -1)
    ##### OUTPUT: Power(t)

#1. Import Library

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.io
import tensorflow as tf
import keras


#2. Data Preparation

feature=pd.read_csv("feature.csv",header=None)
feature.columns=[["Power","Irradiance","Temp","UV","WS","RH"]]
time=pd.read_csv("time.csv",header=None)


time_list=[]
for n in range(len(time)):
    time_sample=str(time.iloc[n,:][0])
    for i in range(len(time.iloc[n,:])-1):
        time_sample=time_sample+str(time.iloc[n,:][i+1])
    time_list.append(time_sample)

seperate_list=[]
```

```
for j in range(len(time_list)):
    seperate=time_list[j].split()
    seperate_list.append(seperate)

time_df=pd.DataFrame(seperate_list)
time_df.columns=[["Date","Time"]]
time_df.to_pickle("time_df.pickle")
time_df.head()

data=pd.concat([time_df,feature],axis=1)
data.head()


#3. Data Pre-Processing

traindata = scipy.io.loadmat('DataSetUnCleaned_Train.mat')
testdata = scipy.io.loadmat('DataSetUnCleaned_Test.mat')
valdata = scipy.io.loadmat('DataSetValidation.mat')

nonlag_trainindex=[]
lag_trainindex=[]
nonlag_testindex=[]
lag_testindex=[]
nonlag_valindex=[]
lag_valindex=[]
for i in range(len(traindata['DataSetUnCleaned_Train']['LaggingIndexOld'][0][0])):
    nonlag_trainindex.append(traindata['DataSetUnCleaned_Train']['LaggingIndexOld'
        ][0][0][i][0])
    lag_trainindex.append(traindata['DataSetUnCleaned_Train']['LaggingIndexOld'
        ][0][0][i][1])

for j in range(len(testdata['DataSetUnCleaned_Test']['LaggingIndexOld'][0][0])):
    nonlag_testindex.append(testdata['DataSetUnCleaned_Test']['LaggingIndexOld'
        ][0][0][j][0])
    lag_testindex.append(testdata['DataSetUnCleaned_Test']['LaggingIndexOld'][0][0][
        j][1])

for k in range(len(valdata['DataSetValidation']['LaggingIndexOld'][0][0])):
    nonlag_valindex.append(valdata['DataSetValidation']['LaggingIndexOld'][0][0][k
        ][0])
    lag_valindex.append(valdata['DataSetValidation']['LaggingIndexOld'][0][0][k][1])

# Minus Index for Python
lag_trainindex=np.array(lag_trainindex).astype('int64')-np.array(1)
nonlag_trainindex=np.array(nonlag_trainindex).astype('int64')-np.array(1)
lag_testindex=np.array(lag_testindex).astype('int64')-np.array(1)
nonlag_testindex=np.array(nonlag_testindex).astype('int64')-np.array(1)
lag_valindex=np.array(lag_valindex).astype('int64')-np.array(1)
nonlag_valindex=np.array(nonlag_valindex).astype('int64')-np.array(1)


# For Training Set
training_set=data[data.index.isin(nonlag_trainindex)].reset_index(drop=True)
training_set.head()
temp_trainlag=data[data.index.isin(lag_trainindex)].Temp.reset_index(drop=True)
irradiance_trainlag=data[data.index.isin(lag_trainindex)].Irradiance.reset_index(
    drop=True)
col_names=['Date','Time','Power','Irradiance','Temp','UV','WS','RH','Irradiance_lag'
    ,'Temp_lag']
training_set=pd.concat([training_set,irradiance_trainlag,temp_trainlag],axis=1)
training_set.columns=col_names
training_set=training_set[['Date','Time','Power','Irradiance','Irradiance_lag','Temp
    ','Temp_lag','UV','WS','RH']]
training_set.head()

```

```
#For Test Set
test_set=data[data.index.isin(nonlag_testindex)].reset_index(drop=True)
test_set.head()
temp_testlag=data[data.index.isin(lag_testindex)].Temp.reset_index(drop=True)
irradiance_testlag=data[data.index.isin(lag_testindex)].Irradiance.reset_index(drop=
    True)
col_names=['Date','Time','Power','Irradiance','Temp','UV','WS','RH','Irradiance_lag'
    ,'Temp_lag']
test_set=pd.concat([test_set,irradiance_testlag,temp_testlag],axis=1)
test_set.columns=col_names
test_set=test_set[['Date','Time','Power','Irradiance','Irradiance_lag','Temp','
    Temp_lag','UV','WS','RH']]
test_set.head()

# For Validation Set
val_set=data[data.index.isin(nonlag_valindex)].reset_index(drop=True)
val_set.head()
temp_vallag=data[data.index.isin(lag_valindex)].Temp.reset_index(drop=True)
irradiance_vallag=data[data.index.isin(lag_valindex)].Irradiance.reset_index(drop=
    True)
col_names=['Date','Time','Power','Irradiance','Temp','UV','WS','RH','Irradiance_lag'
    ,'Temp_lag']
val_set=pd.concat([val_set,irradiance_vallag,temp_vallag],axis=1)
val_set.columns=col_names
val_set=val_set[['Date','Time','Power','Irradiance','Irradiance_lag','Temp','
    Temp_lag','UV','WS','RH']]
val_set.head()


# Normalization
training_set.iloc[:,3:5]=training_set.iloc[:,3:5]/1000
training_set.iloc[:,5:7]=training_set.iloc[:,5:7]/10
test_set.iloc[:,3:5]=test_set.iloc[:,3:5]/1000
test_set.iloc[:,5:7]=test_set.iloc[:,5:7]/10
val_set.iloc[:,3:5]=val_set.iloc[:,3:5]/1000
val_set.iloc[:,5:7]=val_set.iloc[:,5:7]/10

x_train=training_set.iloc[:,3:7]
y_train=training_set.iloc[:,2:3]
x_test=test_set.iloc[:,3:7]
y_test=test_set.iloc[:,2:3]
x_val=val_set.iloc[:,3:7]
y_val=val_set.iloc[:,2:3]


#4. Artificial Neural Networks with Lagging Model

model= keras.Sequential([
                        keras.layers.Dense(3,activation=tf.nn.relu,input_shape=(4,),
                            kernel_initializer=keras.initializers.glorot_uniform(seed
                            =2)),
                        keras.layers.Dense(1)
])


optimizer=tf.train.AdamOptimizer(learning_rate=0.001)
model.compile(loss='mse',optimizer=optimizer,metrics=['mse'])
model.summary()

early_stop = keras.callbacks.EarlyStopping(monitor='mean_squared_error', patience=5)
early_stop
history = model.fit(x_train,y_train, epochs=100,callbacks=[early_stop],
    validation_data=(x_test,y_test))


```

```
#plot MSE
plt.figure()
plt.xlabel('Epoch')
plt.ylabel('Mean Squared Error')
plt.plot(history.epoch, np.array(history.history['mean_squared_error']),
label='Train Loss')
plt.plot(history.epoch, np.array(history.history['val_mean_squared_error']),
label='Validation Loss')
plt.legend()


#plot True Values & predictions

train_predictions = model.predict(x_train).flatten()
test_predictions = model.predict(x_test).flatten()

#plot True Values & predictions
x=[x for x in range(480*1)]
plt.figure(figsize=(20,10))
plt.plot(x,np.array(y_test)[0:480*1,],label="True Values",marker='o')
plt.plot(x,test_predictions[0:480*1,],label="Prediction",marker='o')
plt.legend(['True Value','Prediction'])
plt.ylabel('Power')
plt.xlabel('TimeSample')
plt.title("Artificial Neural Networks with Non-lagging in a day")


mse_train=sum((np.array(y_train.Power).squeeze()-train_predictions)**2)/len(y_train)
mse_test=sum((np.array(y_test.Power).squeeze()-test_predictions)**2)/len(y_test)
print("MSE of Training set:"+"\t"+str(mse_train))
print("MSE of Test set:"+"\t"+str(mse_test))
```

Listing 14: SVR-Datasets.py

```
## 2102531_System Identification Term Project

### Support Vector Regression
#     ##### INPUT:  Irradiance(t) and Temperature(t)
#     ##### OUTPUT: Power(t)


#1. Import Library

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.io
from sklearn.svm import SVR


#2. Data Preparation

feature=pd.read_csv("feature.csv",header=None)
feature.columns=[["Power","Irradiance","Temp","UV","WS","RH"]]
time=pd.read_csv("time.csv",header=None)


time_list=[]
for n in range(len(time)):
    time_sample=str(time.iloc[n,:][0])
    for i in range(len(time.iloc[n,:])-1):
        time_sample=time_sample+str(time.iloc[n,:][i+1])
    time_list.append(time_sample)

seperate_list=[]
```

```
for j in range(len(time_list)):
    seperate=time_list[j].split()
    seperate_list.append(seperate)

time_df=pd.DataFrame(seperate_list)
time_df.columns=[["Date","Time"]]
time_df.to_pickle("time_df.pickle")
time_df.head()

data=pd.concat([time_df,feature],axis=1)
data.head()


#3. Data Pre-Processing

traindata = scipy.io.loadmat('DataSetCleaned_Train.mat')
testdata = scipy.io.loadmat('DataSetCleaned_Test.mat')
valdata = scipy.io.loadmat('DataSetValidation.mat')

nonlag_trainindex=[]
lag_trainindex=[]
nonlag_testindex=[]
lag_testindex=[]
nonlag_valindex=[]
lag_valindex=[]
for i in range(len(traindata['DataSetCleaned_Train']['LaggingIndexOld'][0][0])):
    nonlag_trainindex.append(traindata['DataSetCleaned_Train']['LaggingIndexOld'
        ][0][0][i][0])
    lag_trainindex.append(traindata['DataSetCleaned_Train']['LaggingIndexOld'
        ][0][0][i][1])

for j in range(len(testdata['DataSetCleaned_Test']['LaggingIndexOld'][0][0])):
    nonlag_testindex.append(testdata['DataSetCleaned_Test']['LaggingIndexOld'
        ][0][0][j][0])
    lag_testindex.append(testdata['DataSetCleaned_Test']['LaggingIndexOld'][0][0][j
        ][1])

for k in range(len(valdata['DataSetValidation']['LaggingIndexOld'][0][0])):
    nonlag_valindex.append(valdata['DataSetValidation']['LaggingIndexOld'][0][0][k
        ][0])
    lag_valindex.append(valdata['DataSetValidation']['LaggingIndexOld'][0][0][k][1])


#Minus Index for Python
lag_trainindex=np.array(lag_trainindex).astype('int64')-np.array(1)
nonlag_trainindex=np.array(nonlag_trainindex).astype('int64')-np.array(1)
lag_testindex=np.array(lag_testindex).astype('int64')-np.array(1)
nonlag_testindex=np.array(nonlag_testindex).astype('int64')-np.array(1)
lag_valindex=np.array(lag_valindex).astype('int64')-np.array(1)
nonlag_valindex=np.array(nonlag_valindex).astype('int64')-np.array(1)

# For Training Set
training_set=data[data.index.isin(nonlag_trainindex)].reset_index(drop=True)
training_set.head()

# For Test Set
test_set=data[data.index.isin(nonlag_testindex)].reset_index(drop=True)
test_set.head()

# For Validation Set
val_set=data[data.index.isin(nonlag_valindex)].reset_index(drop=True)
val_set.head()

# Normalization
training_set.iloc[:,3:4]=training_set.iloc[:,3:4]/1000
```

```
training_set.iloc[:,4:5]=training_set.iloc[:,4:5]/10
test_set.iloc[:,3:4]=test_set.iloc[:,3:4]/1000
test_set.iloc[:,4:5]=test_set.iloc[:,4:5]/10
val_set.iloc[:,3:4]=val_set.iloc[:,3:4]/1000
val_set.iloc[:,4:5]=val_set.iloc[:,4:5]/10


x_train=training_set.iloc[:,3:5]
y_train=training_set.iloc[:,2:3]
x_test=test_set.iloc[:,3:5]
y_test=test_set.iloc[:,2:3]
x_val=val_set.iloc[:,3:5]
y_val=val_set.iloc[:,2:3]



#4. Support Vector Regression of PV Model

model=SVR()
from sklearn.model_selection import GridSearchCV
param_grid = {"C":[(10**x) for x in range(0,3)],"epsilon":[0.1,0.5,1]}
grid = GridSearchCV(model,param_grid,cv=5)
grid.fit(x_train,y_train)
grid.best_params_


model=SVR(C=10,epsilon=0.1)
model.fit(x_train,y_train)

train_accuracy=model.score(x_train,y_train)
train_accuracy

test_accuracy=model.score(x_test,y_test)
test_accuracy


from sklearn.metrics import mean_squared_error
MSE_train = mean_squared_error(y_train,model.predict(x_train))
MSE_test = mean_squared_error(y_test,model.predict(x_test))

print("MSE_train:", MSE_train)
print("MSE_test:", MSE_test)
```

Listing 15: SVR-Datasets-NOTCLEAN.py

```
## 2102531_System Identification Term Project

### Support Vector Regression
#     ##### INPUT:  Irradiance(t) and Temperature(t)
#     ##### OUTPUT: Power(t)


#1. Import Library

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.io
from sklearn.svm import SVR


#2. Data Preparation

feature=pd.read_csv("feature.csv",header=None)
feature.columns=[["Power","Irradiance","Temp","UV","WS","RH"]]
```

```
time=pd.read_csv("time.csv",header=None)


time_list=[]
for n in range(len(time)):
    time_sample=str(time.iloc[n,:][0])
    for i in range(len(time.iloc[n,:])-1):
        time_sample=time_sample+str(time.iloc[n,:][i+1])
    time_list.append(time_sample)

seperate_list=[]
for j in range(len(time_list)):
    seperate=time_list[j].split()
    seperate_list.append(seperate)

time_df=pd.DataFrame(seperate_list)
time_df.columns=[["Date","Time"]]
time_df.to_pickle("time_df.pickle")
time_df.head()

data=pd.concat([time_df,feature],axis=1)
data.head()


#3. Data Pre-Processing

traindata = scipy.io.loadmat('DataSetUnCleaned_Train.mat')
testdata = scipy.io.loadmat('DataSetUnCleaned_Test.mat')
valdata = scipy.io.loadmat('DataSetValidation.mat')

nonlag_trainindex=[]
lag_trainindex=[]
nonlag_testindex=[]
lag_testindex=[]
nonlag_valindex=[]
lag_valindex=[]
for i in range(len(traindata['DataSetUnCleaned_Train']['LaggingIndexOld'][0][0])):
    nonlag_trainindex.append(traindata['DataSetUnCleaned_Train']['LaggingIndexOld'
        ][0][0][i][0])
    lag_trainindex.append(traindata['DataSetUnCleaned_Train']['LaggingIndexOld'
        ][0][0][i][1])

for j in range(len(testdata['DataSetUnCleaned_Test']['LaggingIndexOld'][0][0])):
    nonlag_testindex.append(testdata['DataSetUnCleaned_Test']['LaggingIndexOld'
        ][0][0][j][0])
    lag_testindex.append(testdata['DataSetUnCleaned_Test']['LaggingIndexOld'][0][0][
        j][1])

for k in range(len(valdata['DataSetValidation']['LaggingIndexOld'][0][0])):
    nonlag_valindex.append(valdata['DataSetValidation']['LaggingIndexOld'][0][0][k
        ][0])
    lag_valindex.append(valdata['DataSetValidation']['LaggingIndexOld'][0][0][k][1])

# Minus Index for Python
lag_trainindex=np.array(lag_trainindex).astype('int64')-np.array(1)
nonlag_trainindex=np.array(nonlag_trainindex).astype('int64')-np.array(1)
lag_testindex=np.array(lag_testindex).astype('int64')-np.array(1)
nonlag_testindex=np.array(nonlag_testindex).astype('int64')-np.array(1)
lag_valindex=np.array(lag_valindex).astype('int64')-np.array(1)
nonlag_valindex=np.array(nonlag_valindex).astype('int64')-np.array(1)


# For Training Set
training_set=data[data.index.isin(nonlag_trainindex)].reset_index(drop=True)
training_set.head()
```

```
81
82 #For Test Set
83 test_set=data[data.index.isin(nonlag_testindex)].reset_index(drop=True)
84 test_set.head()
85
86 # For Validation Set
87 val_set=data[data.index.isin(nonlag_valindex)].reset_index(drop=True)
88 val_set.head()
89
90 # Normalization
91 training_set.iloc[:,3:4]=training_set.iloc[:,3:4]/1000
92 training_set.iloc[:,4:5]=training_set.iloc[:,4:5]/10
93 test_set.iloc[:,3:4]=test_set.iloc[:,3:4]/1000
94 test_set.iloc[:,4:5]=test_set.iloc[:,4:5]/10
95 val_set.iloc[:,3:4]=val_set.iloc[:,3:4]/1000
96 val_set.iloc[:,4:5]=val_set.iloc[:,4:5]/10
97
98 x_train=training_set.iloc[:,3:5]
99 y_train=training_set.iloc[:,2:3]
100 x_test=test_set.iloc[:,3:5]
101 y_test=test_set.iloc[:,2:3]
102 x_val=val_set.iloc[:,3:5]
103 y_val=val_set.iloc[:,2:3]
104
105
106 #4. Support Vector Regression of PV Model
107
108 model=SVR()
109 from sklearn.model_selection import GridSearchCV
110 param_grid = {"C":[(10**x) for x in range(0,3)],"epsilon":[0.1,0.5,1]}
111 grid = GridSearchCV(model,param_grid,cv=5)
112 grid.fit(x_train,y_train)
113 grid.best_params_
114
115
116 model=SVR(C=10,epsilon=0.1)
117 model.fit(x_train,y_train)
118
119 train_accuracy=model.score(x_train,y_train)
120 train_accuracy
121
122 test_accuracy=model.score(x_test,y_test)
123 test_accuracy
124
125
126 from sklearn.metrics import mean_squared_error
127 MSE_train = mean_squared_error(y_train,model.predict(x_train))
128 MSE_test = mean_squared_error(y_test,model.predict(x_test))
129
130 print("MSE_train:", MSE_train)
131 print("MSE_test:", MSE_test)
```