A

Project Report

on

Solar Radiation Prediction

Submitted in the partial fulfillment of the requirements of Credits for

Bachelor of Technology

in

Intelligent Systems and Control

Submitted by:-

Yashi Baldaniya (2015UEE1440)

Sunayana Gupta (2015UEE1754)

Anudeep Katta Reddy (2015UEE1712)

Supervised by:-

Dr. Rajesh Kumar

Professor

Department of Electrical Engineering

Malaviya National Institute of Technology, Jaipur

November 2018

CONTENTS

ACKNOWLEDGEMENT

It was a privilege to study and work under the guidance of Dr. Rajesh Kumar, Professor, Malaviya National Institute of Technology, Jaipur. We take this opportunity to express our deep sense of gratitude to him for his keen interest, valuable suggestions and constant encouragement throughout the project work in Intelligent Systems and Control. He made us comfortable and free to work due to his relaxed way of supervision.

We are thankful to the Ph.D. scholars for their support and needful guidance. Their suggestions and moral support helped us to complete our work. Last but not the least, we would like to thank our parents for their invaluable support, effort, and unconditional love.

Date : 28 November 2018
Place : MNIT, Jaipur

(Yashi)
(Sunayana)
(Anudeep)

ABSTRACT

The aim of this project is to present the implementation of machine learning techniques in Solar Radiation Prediction. Predicting solar radiation an hour, day, or week ahead is very complex. Traditional modeling techniques such as regression and time series are not robust enough to handle the complexity and the nonlinearity of the predictive variables involved in solar radiation prediction. In this project, we have used Random Forests (RF), an ensemble technique, as an example of a supervised machine learning (SML) technique that is capable of incorporating the complexities and the nonlinearity for predicting solar radiation.

1.Introduction

The need for reliable solar irradiance and solar power forecasting is emerged for the optimal modeling and scheduling of solar photovoltaic power plants. For this purpose, this study conducts an exhaustive and up-to-date review of solar irradiance and solar power forecasting methods used in the literature. Electrical energy systems are very complex and delicate because supply normally has to match demand in order to maintain the stability of the power grid. Most power plants are of the conventional type: fossil-fueled and nuclear power plants. Fossil-fueled power plants use coal, natural gas, and oil to generate electrical energy. Such plants are, however, associated with the emission of greenhouse gases, most notable among them, CO2. Greenhouse gases are known to affect global climate change, which could lead to severe weather conditions, coastal flooding, etc. Consequently, policy-makers have been proposing that a certain amount of renewable energy, solar and wind, be made part of the electrical energy generation portfolio.

One of the biggest challenges of renewable energy resources is predicting their availability and quantity. They are highly variable and subject to daily and annual variations and, therefore, can be difficult to estimate and lead to grid instability. Being able to predict their availability, especially in a day-ahead energy market, would be crucial. Traditional forecasting techniques such as regression, autoregressive integrated moving average (ARIMA), and other time-series models are not robust enough for predicting renewable energy resources.

Traditional machine learning methods used only a single variable. Several variables have to be incorporated for better prediction to be achieved, thereby increasing the dimensionality and making the predictive model much more complex. They also note that solar intensity exhibits complex relationships with weather variables, and that the variables themselves depict complex relationships among each other. All of these complexities in predicting solar radiation, therefore necessitate the use of machine learning techniques.

1.1 Data Preprocessing and Data Visualization

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues. Data preprocessing prepares raw data for further processing.

Data visualization is the graphical representation of information and data. In the world of Big Data, data visualization tools and technologies are essential to analyze massive amounts of information and make data-driven decisions. Data visualization is another form of visual art that grabs our interest and keeps our eyes on the message. Data visualization also helps in easily recognising the missing or corrupt data or unusual trends in the data, if present.

1.2 Machine Learning

Machine learning techniques may be categorized into two main groups; supervised and unsupervised techniques. Supervised machine learning (SML) techniques are used in predictive modeling, where features or attributes or input variables are mapped onto an output or response variable. If the output or response variable is categorical (e.g., yes/no, male/female, etc.) then it is a classification or pattern recognition problem. For an output or response variable that is continuous (nominal, ordinal, or real-valued) then it is a regression problem. SML techniques are the most widely used ML techniques.

Many machine learnings have been used which includes neural networks (NN), support vector machines (SVM), ensemble methods (EM) including random forest (RF) classification and regression, which build a series of trees using a random selection of variables. Artificial neural networks have been used in modeling renewable energy resources such as solar and wind. Furthermore, SVMs have been used to predict wind speed. In this project, we are limited to Linear Regression and specifically, random forest (RF) Regression since this is a regression problem.

1.3 Optimization

An optimization problem consists of maximizing or minimizing a real function by systematically choosing input values from within an allowed set and computing the value of the function. More generally, optimization includes finding "best available" values of some objective function given a defined domain (or input), including a variety of different types of objective functions and different types of domains. Here in this project we have optimised the Random Forest algorithm using Particle Swarm Optimisation.

2. Data Preparation

The dataset used in our project is meteorological data from the HI-SEAS weather station from four months (September through December 2016) between Mission IV and Mission V of NASA.

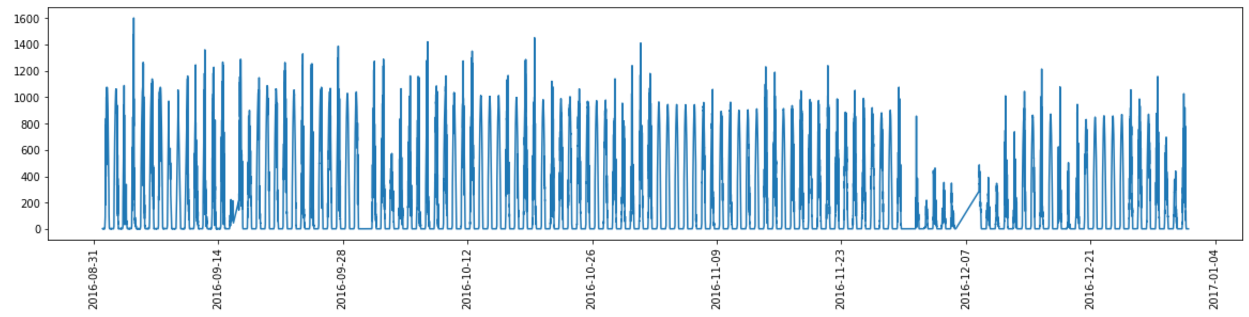
The units of each dataset are:

- Solar radiation: watts per meter^2
- Temperature: degrees Fahrenheit

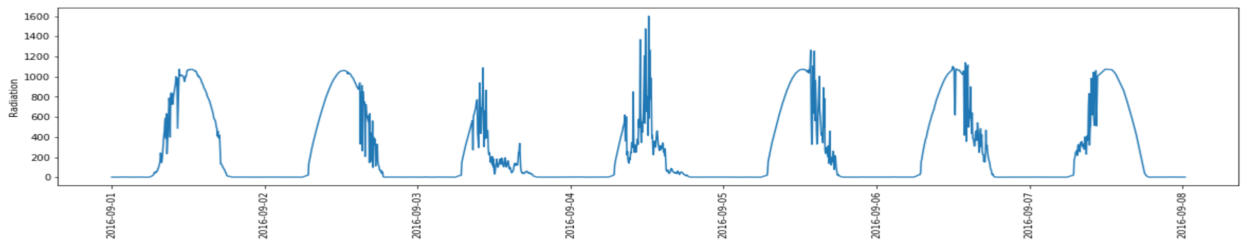
- Humidity: percent
- Barometric pressure: Hg
- Wind direction: degrees
- Wind speed: miles per hour
- Sunrise/sunset: Hawaii time

2.1 Data Preprocessing

The whole data is first visualized to find the occurrence of the missing, corrupt or unusual trend in the data. At the time of visualization, it was evident that a week's data was corrupt which is shown below:

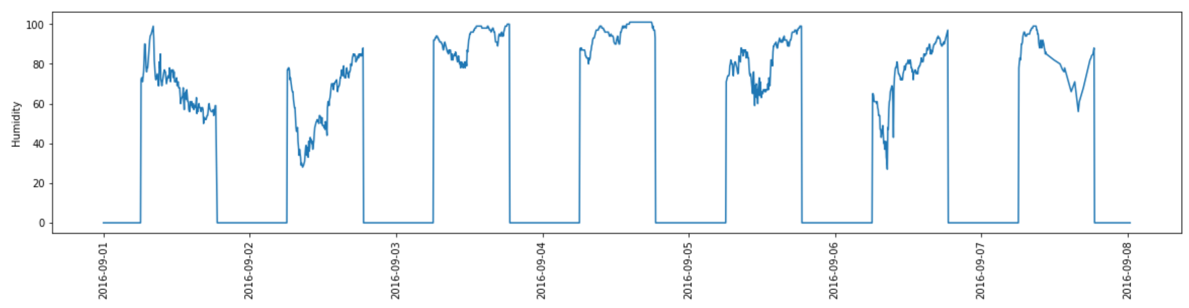


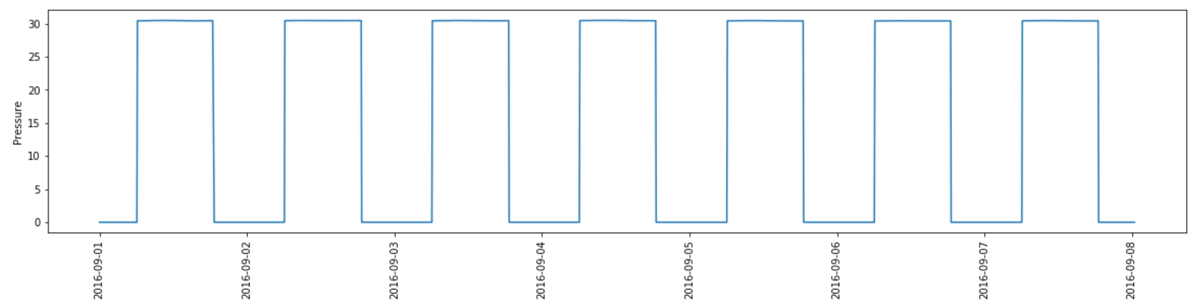
Analysis of the first week's data to understand the trend of solar radiation better :

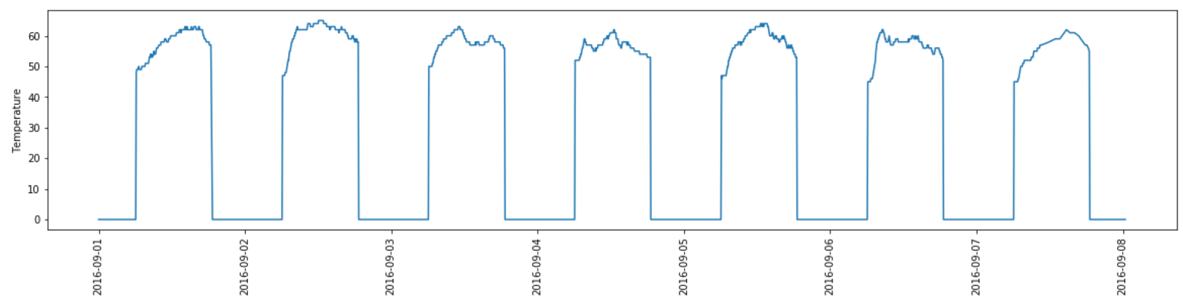


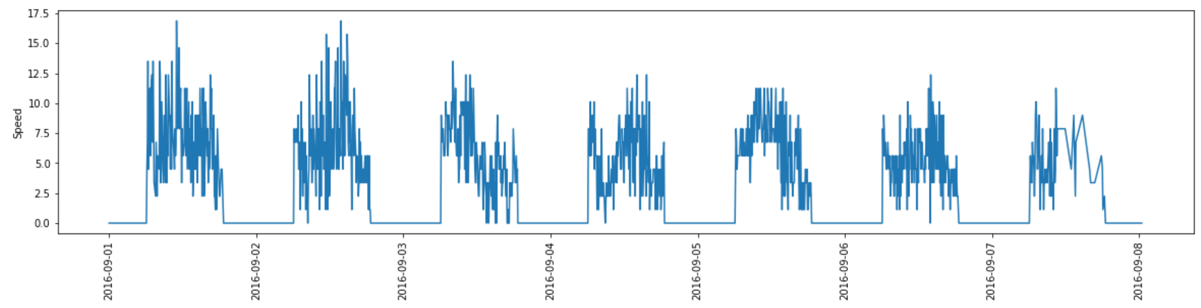
Thus, it is clear from the week's analysis that radiation is zero before sunrise and after sunset, it is obvious as there is no sunlight during these hours. Also, there is one peak value of radiation in a day due to maximum sunlight during peak hours (noon time).

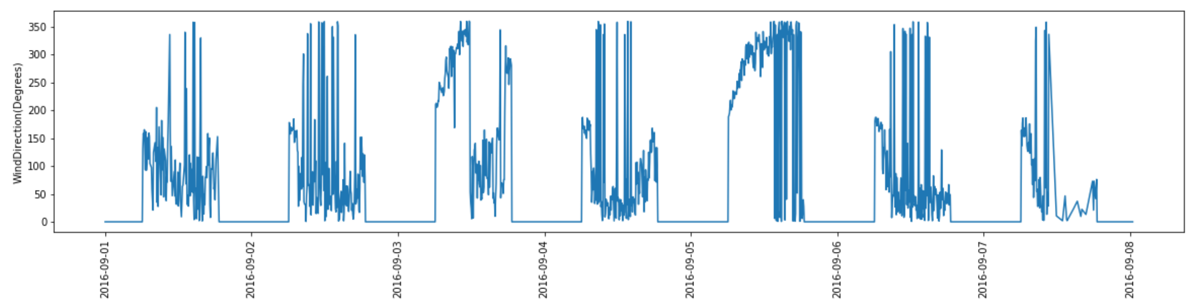
From the inference, we have concluded that there is no effect of any features on radiation when it is zero, so we can make the values of features also to zero for better training of the data.


Humidity
0
20
40
60
80
100
2016-09-01
2016-09-02
2016-09-03
2016-09-04
2016-09-05
2016-09-06
2016-09-07
2016-09-08


Pressure
0
5
10
15
20
25
30
2016-09-01
2016-09-02
2016-09-03
2016-09-04
2016-09-05
2016-09-06
2016-09-07
2016-09-08


Temperature
0
10
20
30
40
50
60
2016-09-01
2016-09-02
2016-09-03
2016-09-04
2016-09-05
2016-09-06
2016-09-07
2016-09-08


Speed
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
2016-09-01
2016-09-02
2016-09-03
2016-09-04
2016-09-05
2016-09-06
2016-09-07
2016-09-08

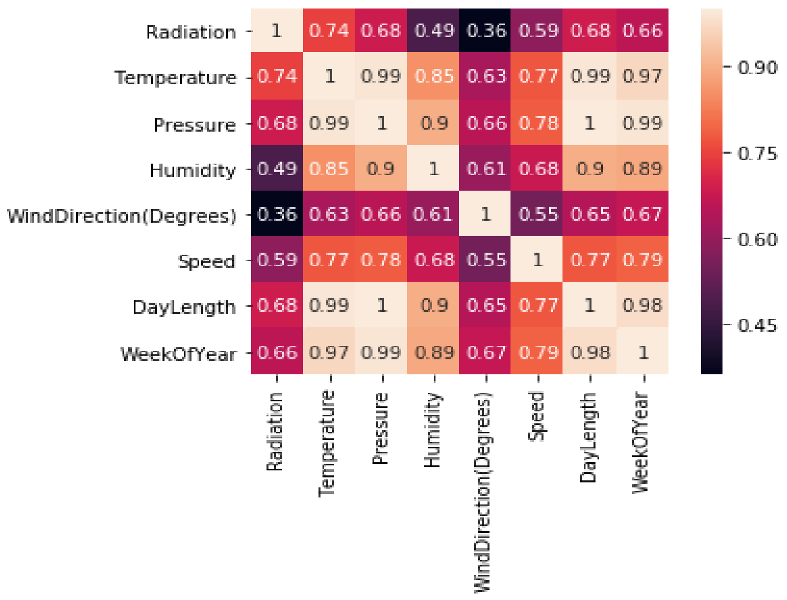

WindDirection(Degrees)
0
50
100
150
200
250
300
350
2016-09-01
2016-09-02
2016-09-03
2016-09-04
2016-09-05
2016-09-06
2016-09-07
2016-09-08

2.2 Correlation

The term "correlation" refers to a mutual relationship or association between quantities. After the analysis of all the features and necessary cleansing of the data, we looked for the correlation of the features (input) with the solar radiation (output) using heatmap and data visualization.

2.2.1 Heatmap

A heat map is a two-dimensional graphical representation of data where the individual values that are contained in a matrix are represented as colors. It is really useful to display a general view of numerical data, not to extract the specific data point.



	Radiation	Temperature	Pressure	Humidity	WindDirection(Degrees)	Speed	DayLength	WeekOfYear
Radiation	1	0.74	0.68	0.49	0.36	0.59	0.68	0.66
Temperature	0.74	1	0.99	0.85	0.63	0.77	0.99	0.97
Pressure	0.68	0.99	1	0.9	0.66	0.78	1	0.99
Humidity	0.49	0.85	0.9	1	0.61	0.68	0.9	0.89
WindDirection(Degrees)	0.36	0.63	0.66	0.61	1	0.55	0.65	0.67
Speed	0.59	0.77	0.78	0.68	0.55	1	0.77	0.79
DayLength	0.68	0.99	1	0.9	0.65	0.77	1	0.98
WeekOfYear	0.66	0.97	0.99	0.89	0.67	0.79	0.98	1

Though both are characteristics of local weather, Wind Direction and Speed , they do not make sense as predictors of radiation. Wind direction and wind direction has moderate correlation with temperature but through engineering judgement we know that this is only *correlation* and not *causation*.

2.2.2 Correlation Visualisation

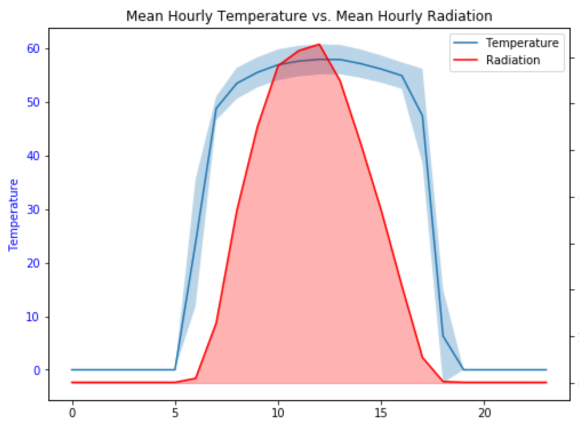
The visualization of highly correlated features with radiation is as shown below-

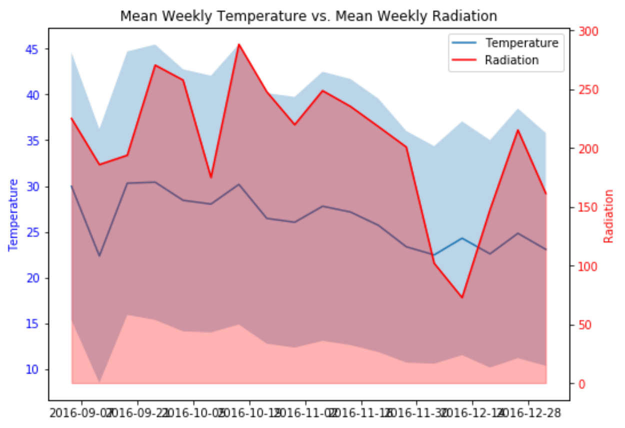
There are three timescales to consider in this dataset:

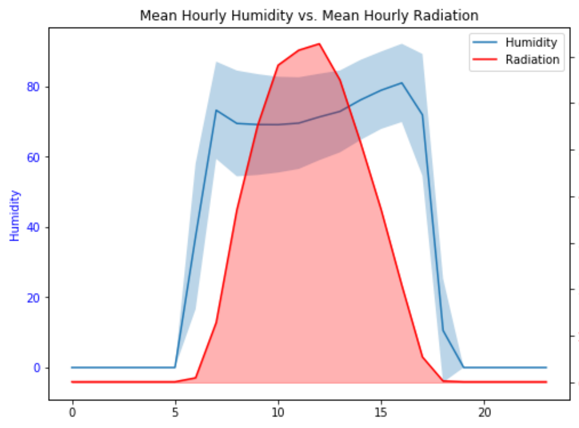
-Monthly
-Daily
-Hourly

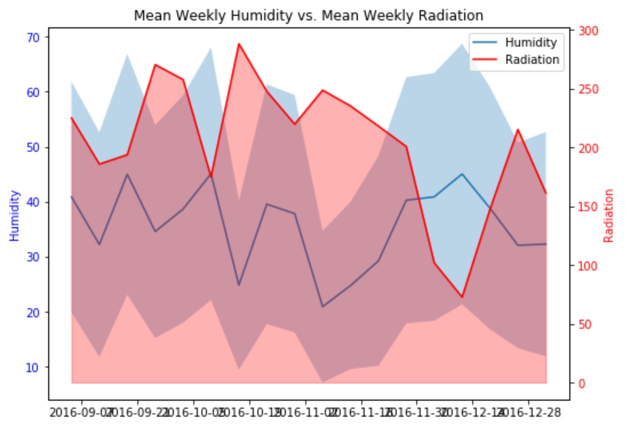
Radiation is expected to vary with the data due to seasonal weather changes. The dataset only contains data from autumn and winter, so the model developed from this data might be less capable of predicting radiation during the summer.

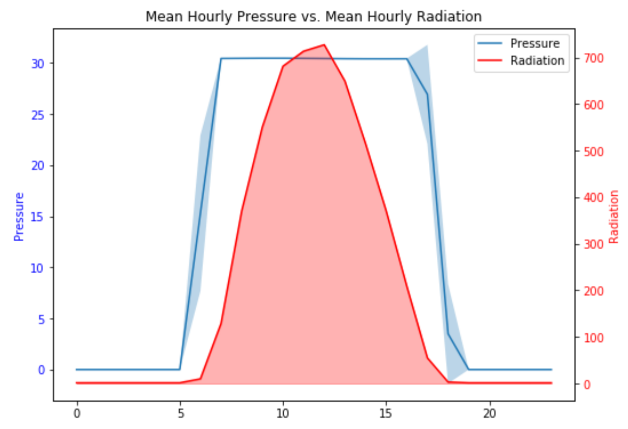
Recalling the most correlated features, we plot Radiation as a function of Temperature, Humidity, and Pressure on the various timescales.

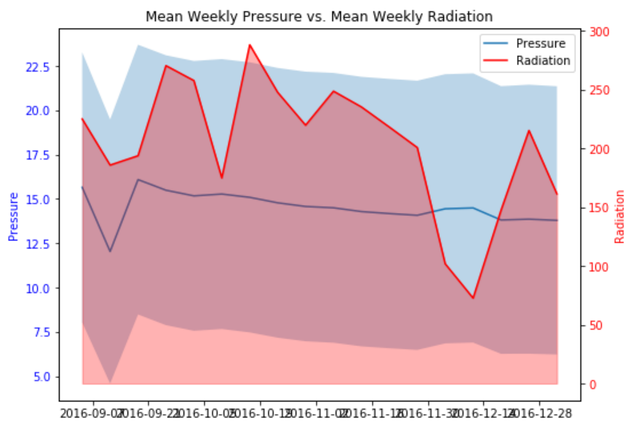












3. Machine Learning Models

We have implemented two machine learning models which are explained below in detail:

3.1 Multivariate Linear Regression

As it is a regression problem, we have first implemented the simplest method of regression i.e Multivariate Linear Regression.

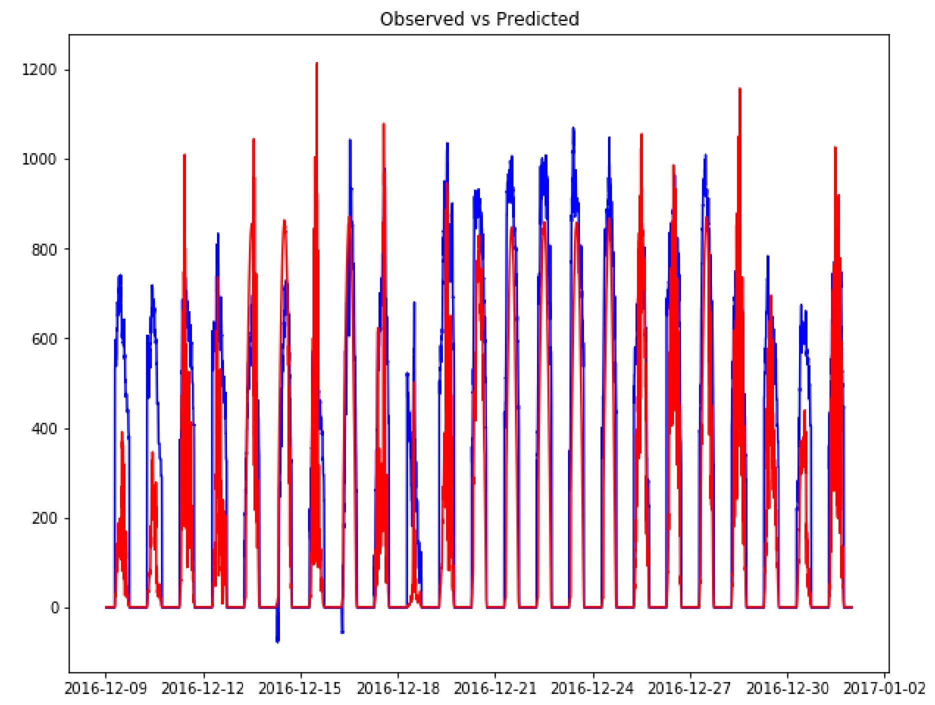
Multiple linear regression attempts to model the relationship between two or more explanatory variables and a response variable by fitting a linear equation to observed data. Every value of the independent variable *x* is associated with a value of the dependent variable *y*.

Formally, the model for multiple linear regression, given *n* observations, is
$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... \beta_p x_{ip} + \varepsilon_i$ for $i = 1,2, ... n$.

In the least-squares model, the best-fitting line for the observed data is calculated by minimizing the sum of the squares of the vertical deviations from each data point to the line (if a point lies on the fitted line exactly, then its vertical deviation is 0). Because the deviations are first squared, then summed, there are no cancellations between positive and negative values. The least-squares estimates b_0, b_1, ... b_p are usually computed by statistical software.

The estimate of the standard error *s* is the square root of the MSE which is minimised.

The result of linear regression is as shown below: (Observed: Red, Predicted: Blue)

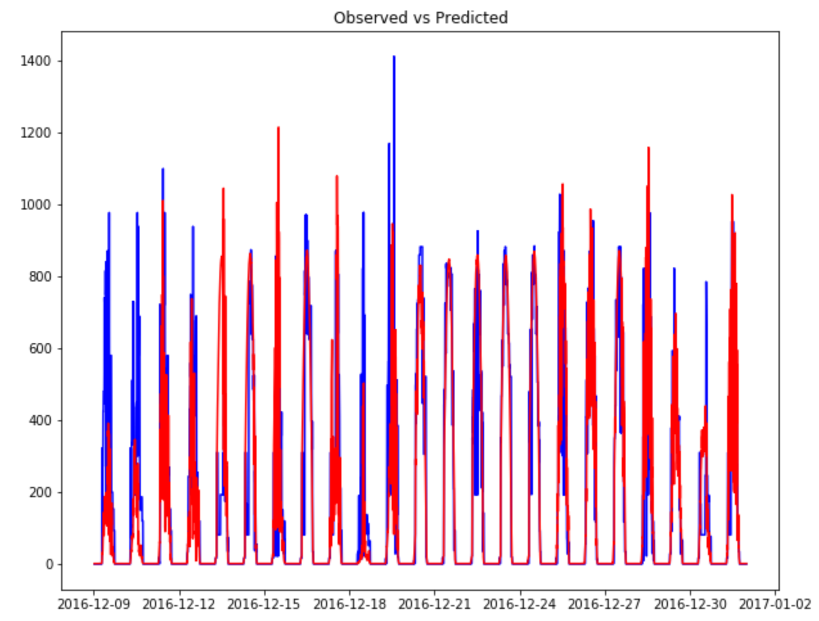


Linear Regression Score : 0.28652080381484146
Mean Absolute Error: 133.55075286498638
RMSE: 223.5623424737968

3.2 Decision Tree Regression:

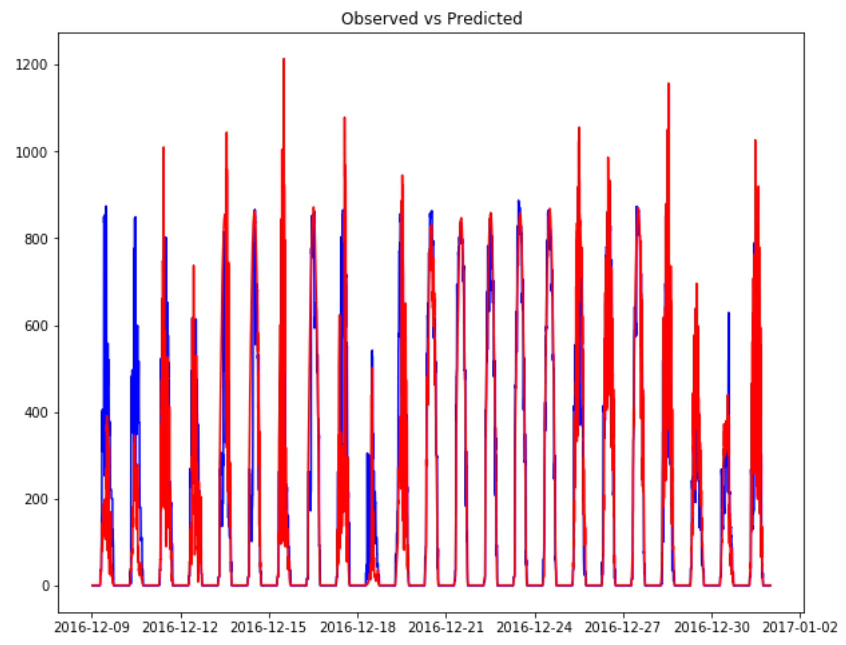
After, Linear Regression we have implemented Decision Tree Regression for better results and better fitting of the data.In Decision Tree Regression, it breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches, each representing values for the attribute tested. Leaf node represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor called root node.

The results are as shown below:
Score: 0.5814205368548427
Mean Absolute Error: 79.7770106355239
RMSE: 171.23674057772365



3.3 Random Forest Regression:

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if bootstrap=True (default). Thus, we have implemented RF regression to overcome overfitting problem and the results are shown below.



Results:
Score: 0.715889913971 11645
Mean Absolute Error: 69.01088710978948
RMSE: 137.05617073908107

In Decision Tree Regression deep decision trees might suffer from overfitting. Random Forest prevents overfitting most of the time, by creating random subsets of the features and building smaller trees using these subsets. Afterwards, it combines the subtrees.

Advantages of RF over other methods:

- Runs efficiently on large databases.
- Handles thousands of input variables without variable deletion.
- Gives estimates of what variables are important in the classification.
- Provides effective methods for estimating missing data and maintains accuracy.
- Avoids overfitting of the model.
- Average prediction of output from the splits performed by the model.

4. Optimization of Random forest:

The Random Forest algorithm which is defined in sklearn package of python runs with certain default parameters when we apply it on a dataset. And these parameters need to be tuned in accordance with a particular dataset for better results. These parameters can be manually changed and then checked for less error but this method is cumbersome and does not provide optimized results.

When the algorithm searches for the splitting criteria, it considers only one variable at a time without considering the potential interaction between other attributes or variables. This approach has a risk of reaching a local optimal solution. To overcome this problem, multivariate splitting criteria are proposed by many researchers. However, the problem of finding the optimal linear split is more difficult and intractable than that of finding the optimal univariate split.

Therefore to make it simpler we used Particle Swarm Optimisation as it simpler, easy to apply and computation burden is also less for this.

4.1 Particle Swarm Optimisation

Particle swarm optimization (PSO) [1] is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. It solves a problem by having a population of candidate solutions, here dubbed particles, and moving these particles around in the search-space according to simple mathematical formulae over the particle's position and velocity. Each particle's movement is influenced by its local best known position, but is also guided toward the best known positions in the search-space, which are updated as better positions are found by other particles. This is expected to move the swarm toward the best solutions.

PSO is a metaheuristic as it makes few or no assumptions about the problem being optimized and can search very large spaces of candidate solutions. However, metaheuristics such as PSO do not guarantee an optimal solution is ever found. Also, PSO does not use the gradient of the problem being optimized, which means PSO does not require that the optimization problem be differentiable as is required by classic optimization methods such as gradient descent and quasi-newton methods.

The concept of particle swarms, although initially introduced for simulating human social behaviors, has become very popular these days as an efficient search and optimization technique. Particle swarm optimization (PSO), does not require any gradient information of the function to be optimized. It uses only primitive mathematical operators and is conceptually very simple.
The canonical PSO model consists of a swarm of particles, which is initialized with a population of random candidate solutions. They move iteratively through the dimension problem space to search the new solutions, where the fitness f can be calculated as the certain qualities measure. Each particle has a position represented by a position-vector x_i(i is the index of the particle) and a velocity represented by a velocity-vector v_i . Each particle remembers its own best position so far in a vector x_i and its j-th dimensional value is x_{ij} . The best position-vector among the swarm so far is then stored in a vector x^* and its j-th dimensional value is x_j*. During the iteration time t, the update of the velocity from the previous velocity to the new velocity is determined, and the new position is then determined by the sum of the previous position and the new velocity.

Below, are the only two equations that make up a bare bones PSO algorithm. As a heads up, “k” references the current iteration, therefore “k+1″ implies the next iteration.

Particle position:
$x_i(k+1) = x_i + v_{i+1}$

Particle velocity:

$v_i(k+1) = w(k)v_i(k)+c_1r_1(p_i(k)-x_i(k))+c_2r_2(p_g(k)-x_i(k))$

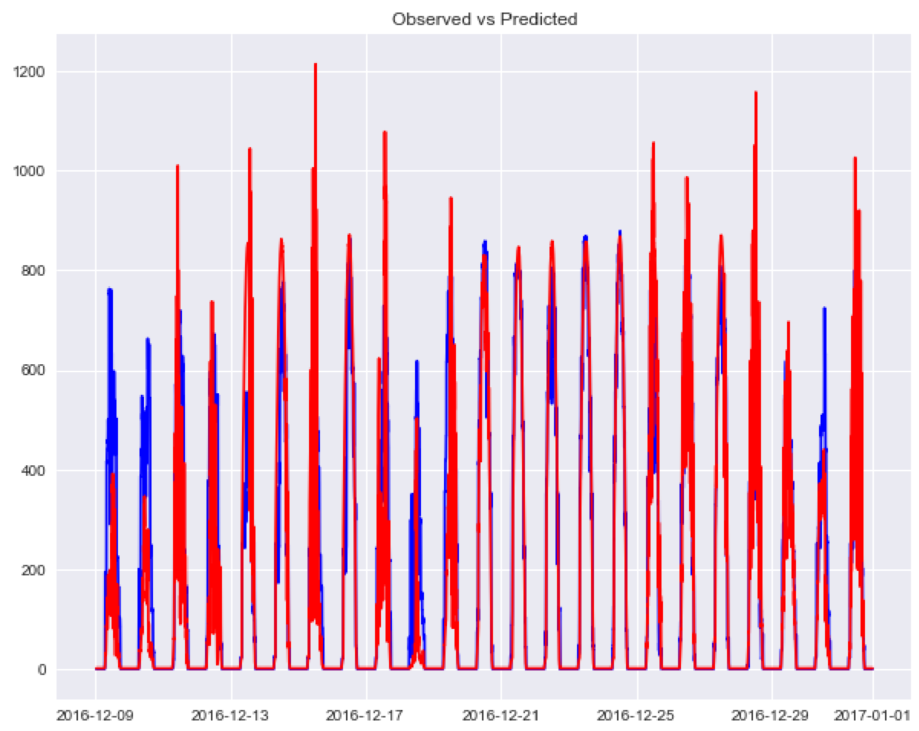
where:

Variable	Definition
$x_i(k)$	particle position
$v_i(k)$	particle position
$p_i(k)$	best individual particle position
$p_g(k)$	best swarm position
$w(k)$	constant inertia weight
c_1,c_2	cognitive and social parameters respectively
r_1,r_2	random numbers between 0 and 1

4.2 Application of PSO in Random Forest

We have used PSO to simultaneously optimise the three most important parameters of Random Forest namely, number of trees(N_estimators), maximum depth of each tree in the forest(max_depth), and maximum number of features to consider while looking for the best split(max_features).

Firstly, we wrote the algorithm for PSO and then we incorporated the random forest inside the cost function and returning the Mean Absolute Error(MAE) from the function to the main function. Hence, this will be the criteria for minimisation and finally we will get the optimised results with minimum MAE. The number of particles taken were 40 and maximum number of iterations were 50. The optimised results are shown below:



Results:

Score: 0.73060760290202298

Mean Absolute Error: 59.67389762547865

RMSE: 137.3727719496423

Conclusion

In this project, we have processed the data and implemented various machine learning models for the prediction of solar radiation using environmental features such as temperature, humidity, pressure, etc. It was found that random forest gave better prediction over other models. Further, optimization of random forest using particle swarm algorithm was used for tuning of parameters of RF regression.

Appendix

Code (in jupyter notebook):

```
import numpy as np
import pandas as pd
import pytz
import datetime
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.svm import SVR
import pickle
from math import sqrt
from sklearn.metrics import mean_squared_error
from sklearn import linear_model
from sklearn.gaussian_process import GaussianProcessRegressor as GPR
from sklearn.gaussian_process.kernels import RBF, ConstantKernel as C
%matplotlib inline

data=pd.read_csv("D:\\ISC Project\\Data\\New Data\\SolarPrediction.csv")
data = data.sort_values(by='UNIXTime', ascending=True).reset_index(drop=True)
HItz= pytz.timezone(zone='US/Hawaii')
datetimeHI = data['UNIXTime'].apply(lambda x:
datetime.datetime.utcfromtimestamp(x).replace(tzinfo=pytz.utc).astimezone(HItz))
data['DatetimeHI'] = datetimeHI

start_train = datetime.datetime(2016, 9, 1).replace(tzinfo=HItz)
end_train = datetime.datetime(2016,12, 5).replace(tzinfo=HItz)
start_test = datetime.datetime(2016,12, 9).replace(tzinfo=HItz)
end_test = datetime.datetime(2016,12, 31).replace(tzinfo=HItz)
def is_day(row):
  sun_rise = datetime.datetime.strptime(row['TimeSunRise'], '%H:%M:%S').time()
  sun_set = datetime.datetime.strptime(row['TimeSunSet'], '%H:%M:%S').time()
  if ((sun_set > row['DatetimeHI'].time()) & (sun_rise < row['DatetimeHI'].time())):
     return 1
  else:
     return 0
```

```
day_bool = np.empty(data.shape[0])

for i in np.arange(data.shape[0]):
  day_bool[i] = is_day(data.iloc[i])

data['Temperature'] = data['Temperature'] * day_bool
data['Pressure'] = data['Pressure'] * day_bool
data['Humidity'] = data['Humidity'] * day_bool
data['WindDirection(Degrees)'] = data['WindDirection(Degrees)'] * day_bool
data['Speed'] = data['Speed'] * day_bool
data['DayLength']=data['DayLength']* day_bool
data['TimeOfDay'] =data['TimeOfDay']* day_bool

data['Radiation'] = data['Radiation'].astype(float)
data['Temperature'] = data['Temperature'].astype(float) # or int
data['Pressure'] = data['Pressure'].astype(float)
data['Humidity'] = data['Humidity'].astype(int) # or int
data['WindDirection(Degrees)'] = data['WindDirection(Degrees)'].astype(float)
data['Speed'] = data['Speed'].astype(float)
data['TimeSunRise'] =  pd.to_datetime(data['TimeSunRise'],format='%H:%M:%S')
data['TimeSunSet'] = pd.to_datetime(data['TimeSunSet'],format='%H:%M:%S')
data['DayLength'] = (data['TimeSunSet']-data['TimeSunRise'])/np.timedelta64(1, 's')
data.drop(['TimeSunRise','TimeSunSet'],axis=1,inplace=True)

plt.figure(figsize=(20, 4))
plt.plot(data['DatetimeHI'],data['Radiation'])
plt.xticks(rotation=90);

plt.figure(figsize=(20, 4))
weekendmarker = datetime.datetime(2016,9, 8).replace(tzinfo=HItz)
weekonedata = data[data['DatetimeHI'] < weekendmarker]
plt.plot(weekonedata['DatetimeHI'], weekonedata['Radiation'])
plt.xticks(rotation=90);
plt.ylabel('Radiation')

cols=['Radiation','Humidity','Pressure','Temperature','WindDirection(Degrees)','Speed']
for x in cols:
   weekendmarker = datetime.datetime(2016,9, 8).replace(tzinfo=HItz)
```

```
    weekonedata = data[data['DatetimeHI'] < weekendmarker]
    plt.figure(figsize=(20, 4))
    plt.plot(weekonedata['DatetimeHI'], weekonedata[x])
    plt.xticks(rotation=90);
    plt.ylabel(x)

data.set_index('DatetimeHI',inplace=True)
data['WeekOfYear'] = data.index.week
data['WeekOfYear']=data['WeekOfYear']* day_bool
data.drop(['UNIXTime'],axis=1,inplace=True)
corr = data.corr()
sns.heatmap(corr, square=True,annot=True)

def color_y_axis(ax, color):
    '''Color y axis on two-axis plots'''
    for t in ax.get_yticklabels():
        t.set_color(color)
    ax.yaxis.label.set_color(color)
    return None
def plotVs(df,timescale,feature1,feature2,ax1):
    '''Plot feature vs radiation'''
    ax2=ax1.twinx()
    df_grouped= df.groupby(timescale)

    df_feature1 = df_grouped[feature1].mean()
    df_feature1_errorpos =  df_feature1+df_grouped[feature1].std()/2
    df_feature1_errorneg =  df_feature1-df_grouped[feature1].std()/2
    ax1.plot(df_feature1)
    ax1.fill_between(df_feature1.index, df_feature1_errorpos.values, df_feature1_errorneg.values, 
alpha=0.3, antialiased=True)
    ax1.set_ylabel(feature1)
    color_y_axis(ax1, 'b')
    if feature2 == 'Radiation':
        rad = df_grouped['Radiation'].mean()
        ax2.plot(rad,'r')
        ax2.fill_between(df_feature1.index, 0, rad, alpha=0.3, antialiased=True, color='red')
        ax2.set_ylabel('Radiation')
        color_y_axis(ax2, 'r')
    else:
        df_feature2 = df_grouped[feature2].mean()
        df_feature2_errorpos =  df_feature2+df_grouped[feature2].std()/2
        df_feature2_errorneg =  df_feature2-df_grouped[feature2].std()/2
        ax1.plot(df_feature2)
        ax1.fill_between(df_feature2.index, df_feature2_errorpos.values,
df_feature2_errorneg.values, alpha=0.3, antialiased=True)
        ax1.set_ylabel(feature2)
        color_y_axis(ax1, 'g')
    return ax1, ax2
def HourlyWeeklyVs(df,feature1,feature2):
    '''Plot a feature vs radiation for time of day and week of year'''
    plt.figure(figsize=(18, 6))
    ax=plt.subplot(121) # hourly
    ax1,ax2 = plotVs(df,df.index.hour,feature1,feature2,ax)
    lines1, labels1 = ax1.get_legend_handles_labels()
    lines2, labels2 = ax2.get_legend_handles_labels()
    ax2.legend(lines1 + lines2, labels1 + labels2)
    plt.xlabel('Hour of Day (Local Time)')
    plt.title('Mean Hourly {0} vs. Mean Hourly {1}'.format(feature1,feature2))

    ax=plt.subplot(122) # weekly
    ax1, ax2 = plotVs(df,pd.Grouper(freq='W'),feature1,feature2,ax)
    lines1, labels1 = ax1.get_legend_handles_labels()
    lines2, labels2 = ax2.get_legend_handles_labels()
    ax2.legend(lines1 + lines2, labels1 + labels2)
    plt.xlabel('Week of Year')
    plt.title('Mean Weekly {0} vs. Mean Weekly {1}'.format(feature1,feature2))
    return

feature_list=['Radiation','Temperature','Humidity','Pressure']
for feature in feature_list[1:]: # radiation vs feature
    HourlyWeeklyVs(data,feature,feature_list[0])
    plt.xticks(rotation=90);
    plt.show()
    plt.ylabel(feature)

data.drop(['WindDirection(Degrees)','Speed','Data','Time'], axis=1, inplace=True)
data['TimeOfDay'] = data.index.haour
```

```
plt.figure(figsize=(20, 4))
marker1 = datetime.datetime(2016,12, 5).replace(tzinfo=HItz)
marker2 = datetime.datetime(2016,12, 9).replace(tzinfo=HItz)
errordata1 = data[(data.index > marker1) & (data.index < marker2) ]
plt.plot( errordata1['Radiation'] )
plt.xticks(rotation=90);

data['DatetimeHI']=data.index
TRAIN = data[(data.index < marker1)]
TEST = data[(data.index > marker2)]

for ix in TRAIN.index:
    if (TRAIN.loc[ix, 'Temperature'] == 0):
        TRAIN.loc[ix,'Radiation']=0

X_train = TRAIN.drop(['Radiation','DatetimeHI'],axis=1)
X_test = TEST.drop(['Radiation','DatetimeHI'],axis=1)
y_train = TRAIN['Radiation']
y_test = TEST['Radiation']

lin_reg = linear_model.LinearRegression()
lin_reg.fit(X_train, y_train)
lin_reg.score(X=X_test, y=y_test)
y_pred = lin_reg.predict(X_test)
plt.figure(figsize=(10, 8))
plt.plot(TEST['DatetimeHI'] ,y_pred, c='blue', label='Predicted')
plt.plot( TEST['DatetimeHI'],y_test, c='red', label='Observed')
plt.title('Observed vs Predicted')
rms = sqrt(mean_squared_error(y_test, y_pred))
rms
error=abs(y_pred-y_test)
mae=np.mean(error)
mae

from sklearn.tree import DecisionTreeRegressor as DTR
dt = DTR()
dt.fit(X_train, y_train)
dt.score(X=X_test, y=y_test)
y_pred = dt.predict(X_test)
```

```
plt.figure(figsize=(10, 8))
plt.plot(TEST['DatetimeHI'], y_pred, c='blue', label='Predicted')
plt.plot(TEST['DatetimeHI'], y_test, c='red', label='Observed')
plt.title('Observed vs Predicted')
rms = sqrt(mean_squared_error(y_test, y_pred))
rms
error=abs(y_pred-y_test)
mae=np.mean(error)
mae

from sklearn.ensemble import RandomForestRegressor as RFR
random_forest = RFR(bootstrap=True, criterion='mae')
random_forest.fit(X_train, y_train)
random_forest.score(X=X_test, y=y_test)
y_pred = random_forest.predict(X_test)
plt.figure(figsize=(10, 8))
plt.plot(TEST['DatetimeHI'], y_pred, c='blue', label='Predicted')
plt.plot(TEST['DatetimeHI'], y_test, c='red', label='Observed')
plt.title('Observed vs Predicted')
rms = sqrt(mean_squared_error(y_test, y_pred))
rms
error=abs(y_pred-y_test)
mae=np.mean(error)
mae

from __future__ import division
import random
import math

def func1(x):
    for i in range(len(x)):
        train_results = []
        test_results = []
        rf = RFR(n_estimators=int(x[i]),max_depth=int(x[i+1]),
n_jobs=-1,max_features=int(x[i+2]))
        rf.fit(X_train, y_train)
        train_pred = rf.predict(X_train)
        errors1=abs(train_pred-y_train)
        mae1=np.mean(errors1)
        y_pred = rf.predict(X_test)
        errors2=abs(y_pred-y_test)
        mae2=np.mean(errors2)
        return mae2

class Particle:
    def __init__(self,x0):
        self.position_i=[]          # particle position
        self.velocity_i=[]          # particle velocity
        self.pos_best_i=[]          # best position individual
        self.err_best_i=-1          # best error individual
        self.err_i=-1               # error individual

        for i in range(0,num_dimensions):
            self.velocity_i.append(random.randint(-1,1))
            self.position_i.append(x0[i])

      def evaluate(self,costFunc):
        self.err_i=costFunc(self.position_i)


        if self.err_i<self.err_best_i or self.err_best_i==-1:
            self.pos_best_i=self.position_i
            self.err_best_i=self.err_i



    def update_velocity(self,pos_best_g):
        w=1       # constant inertia weight
        c1=1       # cognative constant
        c2=2       # social constant

        for i in range(0,num_dimensions):
            r1=random.random()
            r2=random.random()

            vel_cognitive=c1*r1*(self.pos_best_i[i]-self.position_i[i])
            vel_social=c2*r2*(pos_best_g[i]-self.position_i[i])
            self.velocity_i[i]=w*self.velocity_i[i]+vel_cognitive+vel_social
    def update_position(self,bounds):
        for i in range(0,num_dimensions):
            self.position_i[i]=self.position_i[i]+self.velocity_i[i]


            if self.position_i[i]>=bounds[i][1]:
                self.position_i[i]=bounds[i][1]


            if self.position_i[i]<=bounds[i][0]:
                self.position_i[i]=bounds[i][0]

class PSO():
    def __init__(self,costFunc,x0,bounds,num_particles,maxiter):
        global num_dimensions

        num_dimensions=len(x0)
        err_best_g=-1                   # best error for group
        pos_best_g=[]                   # best position for group


        swarm=[]
        for i in range(0,num_particles):
            swarm.append(Particle(x0))

        # Optimization loop
        i=0
        while i<maxiter:


            for j in range(0,num_particles):
                swarm[j].evaluate(costFunc)


                if swarm[j].err_i<err_best_g or err_best_g==-1:
                    pos_best_g=list(swarm[j].position_i)
                    err_best_g=float(swarm[j].err_i)
            for j in range(0,num_particles):
                swarm[j].update_velocity(pos_best_g)
                swarm[j].update_position(bounds)
            i+=1


        print('FINAL:')
        print(pos_best_g)
        print(err_best_g)

if __name__ == "__PSO__":
    main()

initial=[5,1,1]               # initial starting location [x1,x2...]
bounds=[(1,200),(1,64),(1,6)] # input bounds [(x1_min,x1_max),(x2_min,x2_max)...]
PSO(func1,initial,bounds,num_particles=40,maxiter=50)
```

Bibliography

[1] Frank E. Yeboah, Robert Pyle, and Christian A. Bock Hyeng, "Predicting Solar Radiation For Renewable Energy Technologies: A Random Forest Approach," *ResearchGate*, January 2015.

[2] Kuk Yeol Bae ,Han Seung Jang, and Dan Keun Sung,"Hourly Solar Irradiance Prediction Based on Support Vector Machine and Its Error Analysis", IEEE Transactions On Power Systems, Vol. 32, No. 2, March 2017.

[3] Mehmet Yesilbudak, Medine Colak, Ramazan Bayindir, "What are the Current Status and Future Prospects in Solar Irradiance and Solar Power Forecasting?", International Journal of Renewable Energy Research, Vol.8, No.1, March, 2018.

[4] Chi-Hyuck Jun, Yun-Ju Cho, Hyeseon Lee, "Improving Tree-Based Classification Rules Using a Particle Swarm Optimization", HAL archives-ouvertes, Sep 2012.

[5] Dataset: https://2017.spaceappschallenge.org/challenges/earth-and-us/you-are-my-sunshine

[6] Jae-Gon Kim1 , Dong-Hyuk Kim1, Woo-Sik Yoo1, Joung-Yun Lee1, Yong Bae Kim2 , "Daily prediction of solar power generation based on weather forecast information in Korea", IET Renew. Power Gener., 2017, Vol. 11 Iss. 10.