National Institute of Technology, Tiruchirappalli

Summer Internship Program 2019 under AICTE-Margdarshan scheme

Machine Learning Algorithms for Identification of Solar PV faults

Siddharth Prince 113116104101

Vel Tech Multi Tech Dr.Rangarajan Dr.Sagunthala Engineering College

June 2019

BONAFIDE CERTIFICATE

This is to certify that the internship titled ‘Machine Learning Algorithms for Identification of Solar PV faults‘ is a bonafide record of the work done by

Siddharth Prince (113116104101)

in partial fulfillment of the requirements for Internship Program in **Computer Science and Engineering Department** of the **NATIONAL INSTITUTE OF TECHNOLOGY, TIRUCHIRAPPALLI**, during May-June 2019.

Name and Signature of the Mentor

Coordinator

AICTE-Margdarshan scheme

Abstract

Fault detection in photovoltaic plants is essential to guarantee their reliability, safety, and to maximize operating profitability and avoid expensive maintenance. It is difficult to detect the faults by conventional protection devices, leading to safety issues and fire hazards in PV fields. So, machine learning techniques have been proposed for fault detection based on measurements, such as PV array voltage, current, power and irradiance. Classification is a technique that helps to classify the data points in a dataset into 2 (binary classification) or more (multinomial classification) classes. Different classification algorithms are used to identify the temporary and permanent PV faults. In supervised learning the data has to be labelled into appropriate classes so that the model can be trained. The models are trained with classification algorithms like Naive Bayes, Logistic Regression, Decision Trees used to identify the line to line fault. The ensemble algorithms like random forest and Gradient boosting method are used to get more accurate solutions. The comparison among the accuracy of classification algorithms is validated. The fault detection method is successfully integrated with LabVIEW using Data Acquisition system using which the voltage, current and irradiance readings are monitored and recorded. Experimental results over real data from a 1.3kWp grid-connected plant, shows the superior detection of PV faults using Classification techniques while being able to successfully differentiate from partial shading conditions.

Acknowledgements

I express my sincere gratitude to **Dr. M.Brindha, Assistant Professor,** Department of Computer Science and Engineering, National Institute of Technology, Tiruchirappalli under whose timely guidance and valuable inputs, I was able to successfully complete my project successfully.

I express my heartfelt thanks to **Dr. Rajeswari Sridhar, HOD,** Department of Computer Science and Engineering, National Institute of Technology, Tiruchirappalli for kindly allowing me to avail the facilities of the department during the internship.

I express my gratitude to the research scholars, **Ms. T. Janani** from the Department of Computer Science and Engineering, National Institute of Technology, Tiruchirappalli**, Mrs. C. Sowthily Kaviarasu** and **Mr. B. Pradeep Kumar** from the Department of Electrical and Electronics Engineering, National Institute of Technology, Tiruchirappalli for all their support which helped me in making my project work a success.

I also thank **the Department of Electrical and Electronics Engineering**, National Institute of Technology, Tiruchirappalli for allowing and helping me to take readings from the Solar PV System present there.

I thank **the Department of Computer Science and Engineering,** Vel Tech Multi Tech Dr. Rangarajan Dr. Sagunthala Engineering College, Avadi for choosing me as the candidate from the department for the Summer Internship programme under the Margdarshan scheme.

I would finally like to express my sincere gratitude to AICTE for providing this opportunity under the Margdarshan Scheme.

Siddharth Prince

Table of contents

Abstract

Acknowledgements

Table of contents

CHAPTER 1 **1**

1. Introduction 1

1.1 Types of Faults in PV System 1

1.2 Various conditions in PV system 3

1.2.1 Normal Condition 3

1.2.2 Temporary faults 4

1.2.3 Permanent faults 5

1.2.3.1 Short-circuit faults 5

1.3 Machine Learning 7

1.3.1 Basic Classification: 7

1.3.1.1 Naive Bayes 7

1.3.1.2 Logistic Regression 8

1.3.1.3 Decision Trees 9

1.3.2 Ensemble techniques 9

1.3.2.1 Random forests 9

1.3.2.2 Gradient Boosting Machine 9

CHAPTER 2 **10**

2. Literature Survey 10

CHAPTER 3 **11**

3. Methodology and Results 11

3.1 Experimental setup specifications 11

3.1.1 Simulink panel model specification 11

3.1.2 Simulink array specification 11

3.2 Data Acquisition and Labelling 12

3.2.1 Experimental setup 13
3.2.2 Data preprocessing and labelling 13
3.3 Training, testing and results 14
3.3.1 Data analysis 14
3.3.2 Training and testing 16
3.3.3 Results 16
Appendix **18**
Sample Code 18

CHAPTER 1

1. Introduction

With Global Warming and Climate Change on the rise, it is imperative than ever to move to cleaner sources of energy. Solar energy is one of the most prominent sources of renewable energy. Harvesting solar energy using photovoltaic (PV) arrays is advantageous for a number of reasons, few of them being easy availability, long life and easy maintenance of the system.

The power output of PV systems is dependent on various factors like the intensity of sunlight (irradiance) that falls on it, temperature, environmental conditions and failures/faults that may occur during its operation. The dependence on irradiation means that sudden shading of any module will cause a corresponding sudden drop in the power readings. Short circuiting of modules cause internal fault known as Line-Line (LL) fault, it will also cause a sudden drop in the power output. This leads to ambiguity while monitoring and may indicate drop in power due to shading as an internal LL fault.

1.1 Types of Faults in PV System

Faults reduces the output power and the performance of the PV system. Faults can also be dangerous and cause fires. Faults can occur due to (a) faults in utility grid, (b) faults in power converters (c) faults in PV arrays. Of these, the most hardest to detect are, faults that occur in PV arrays. These consist of short-circuit faults which are Line-Line (LL) and Line-Ground (LG) faults. This project focuses on detecting the LL faults and successfully differentiating them from temporary faults (external disturbances like falling of shade or cloud passing etc.) thus not raising any alarm when the temporary fault occurs, which has very similar characteristics to the LL faults. Fig 1. below represents the types of PV Array faults that can occur.

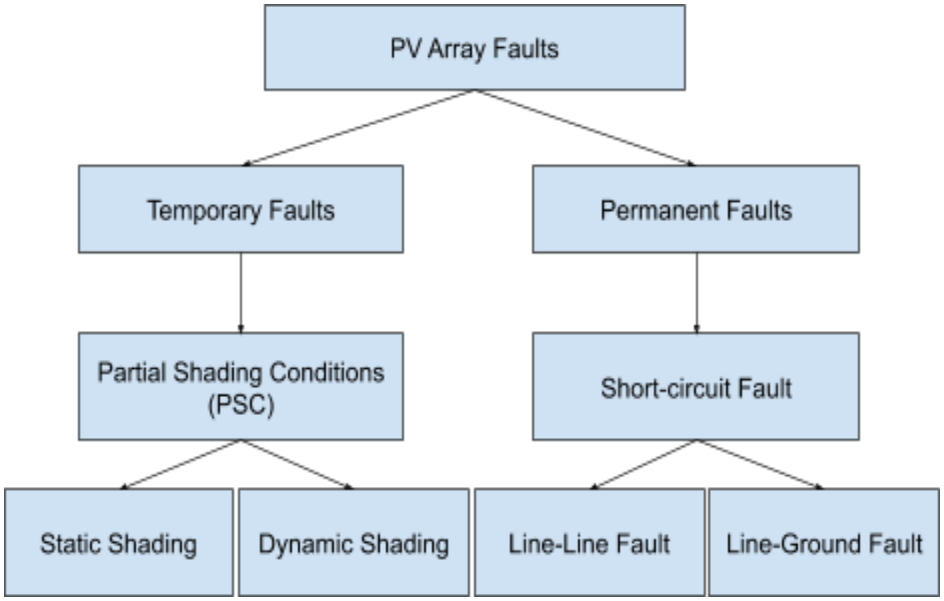

Fig 1. Types of PV Array Faults

The design of the PV array is done using one-diode model whose equation is represented in (1) and the circuit is represented by Fig 2.

$$I_{PVm} = I_{ph} - I_s[exp(\frac{V_{PVm}+I_{PVm}*R_S}{A*Vt}) - 1] - \frac{V_{PVm}+I_{PVm}*R_S}{R_{sh}} \quad (1)$$

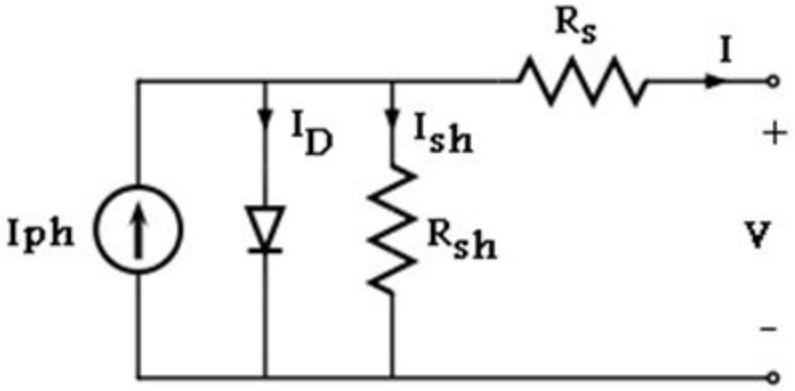

Fig 2. One diode model

1.2 Various conditions in PV system

1.2.1 Normal Condition

Under normal conditions, the PV system has no short circuits nor any partial shading as indicated by Fig 3 (a). The corresponding voltage vs current and voltage vs power distributions are plotted in Fig 3 (b) and Fig 3 (c) respectively.

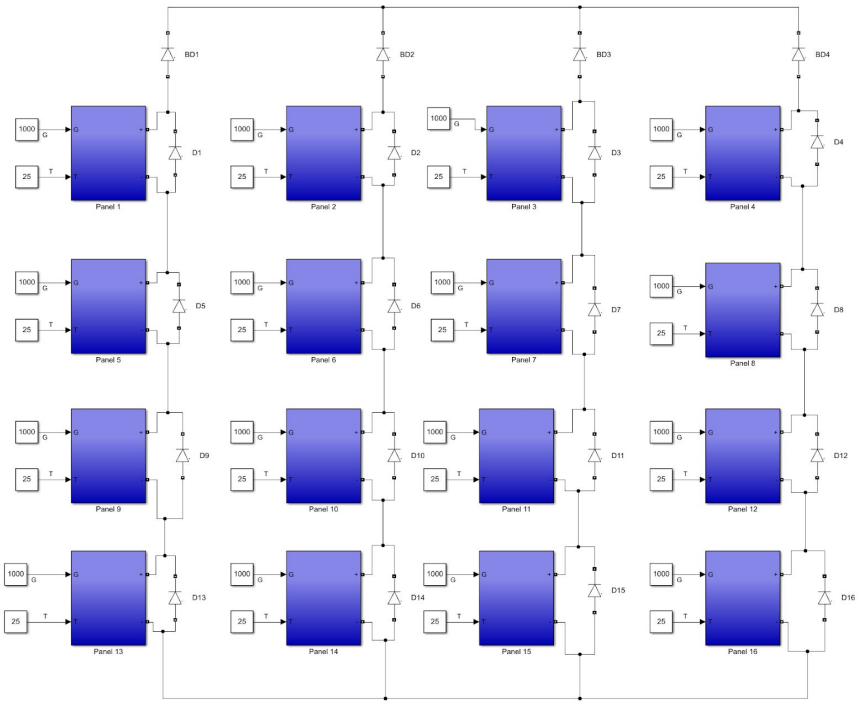

Fig 3 (a). Normal condition

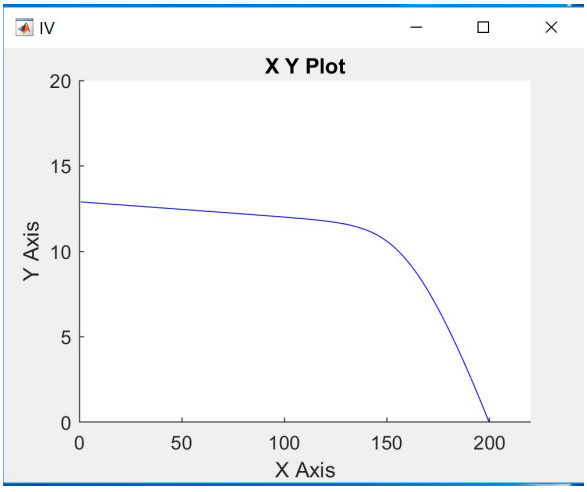

Fig 3 (b). Voltage Vs Current- Normal condition

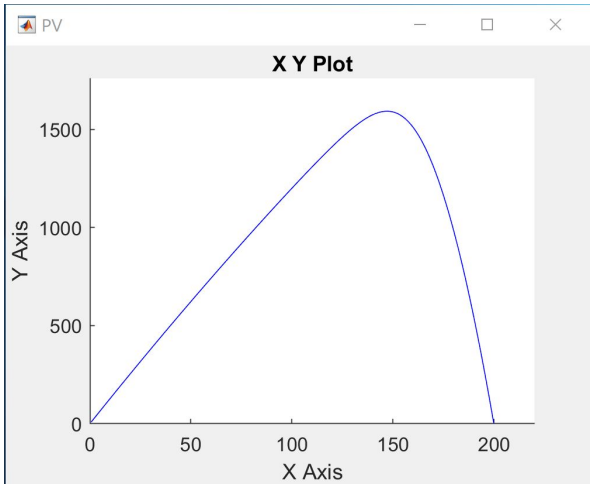

Fig 3 (c). Voltage Vs Power -Normal condition

1.2.2 Temporary faults

Temporary faults occur because of shading on the PV modules due to objects in the environment like buildings, lamp posts, trees, clouds, dirt and other objects that block the sunlight. Partial shading is generally considered as a temporary fault. They are also of two types, static shading where the shading is caused by objects which get stuck on the panels like bird droppings, dirt, snow, etc. and dynamic shading where shading is caused by moving shadows like clouds, trees, buildings, etc. where the shading area varies with time. The Fig 4 (a) shows partial shading in one module of the PV system while Fig 4 (b) and Fig 4 (c) shows the corresponding voltage vs current and voltage vs power distributions as a result of partial shading on one module.

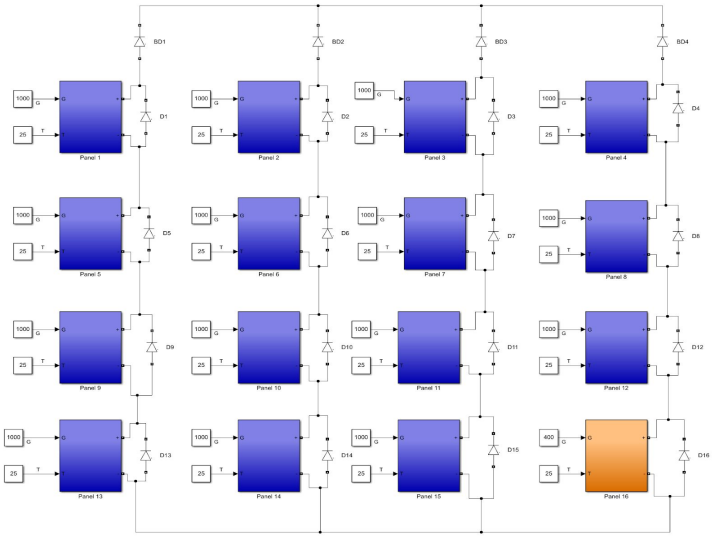

Fig 4 (a). Partial shading

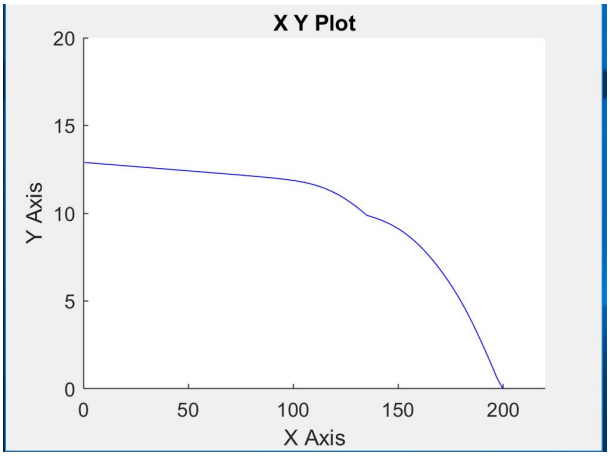

Fig 4 (b). Voltage Vs Current - Partial Shading

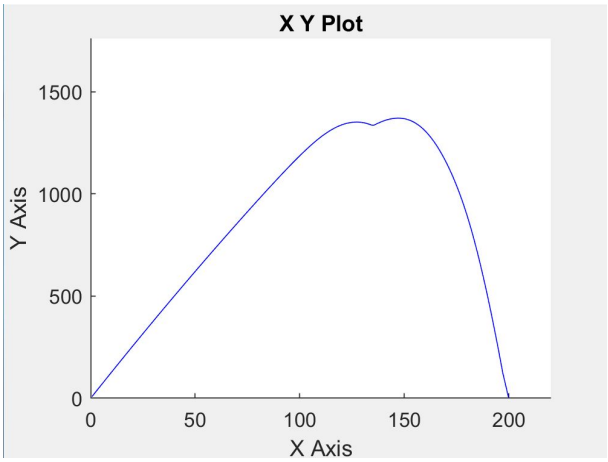

Fig 4 (c). Voltage Vs Power - Partial Shading

1.2.3 Permanent faults

1.2.3.1 Short-circuit faults

A short-circuit fault can be defined as a bridging fault between two points in a PV array system. They are of two types,

(a) Line-Line fault

A Line-Line fault is found to happen when an unmeant low impedance current flowing path occurs between two points of the same or adjacent strings at different potentials in a PV system. It is mainly caused in PV systems due to insulation failures of cables such as chewing of cables by rodents, mechanical damage, water ingression or corrosion. One module and two module mismatch is represented by Fig 5 (a) and Fig 6 (a) respectively and their corresponding voltage vs current and voltage vs power distributions are plotted in Fig 5 (b), Fig 5 (c), Fig 6 (b) and Fig 6 (c) respectively.

(b) Line-Ground fault

A Line-Ground fault occurs due to an unintentional current patch of low impedance between one of the current carrying conductors and the ground/earth. Similar reasons such as corrosion, aging or mechanical damage. If a ground fault remains undetected, it may generate a DC arc within the fault which may lead to a large fire accident. This however can be detected by ground fault protection devices (GFPD) and Arc Fault Circuit Interrupters (AFCI) and is present in majority of PV systems.

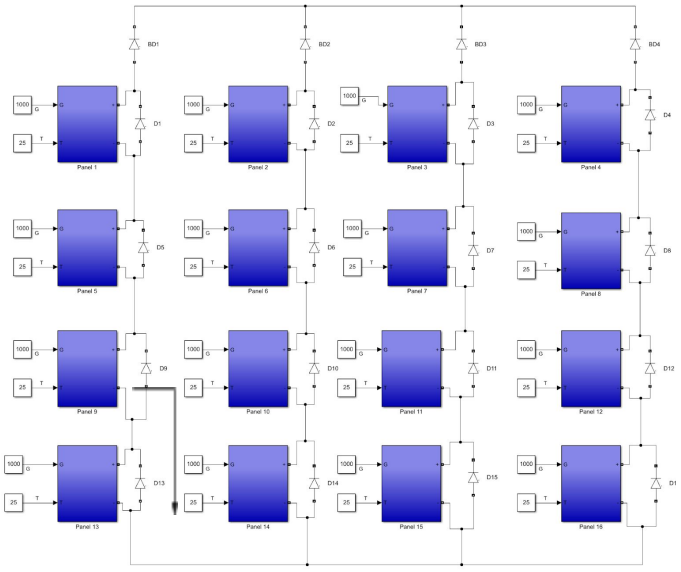

Fig 5 (a). Line-Line Fault One module mismatch

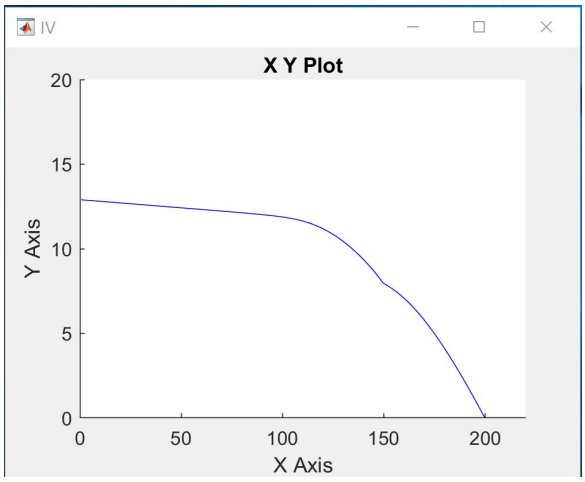

Fig 5 (b). Voltage Vs Current - One module mismatch mismatch

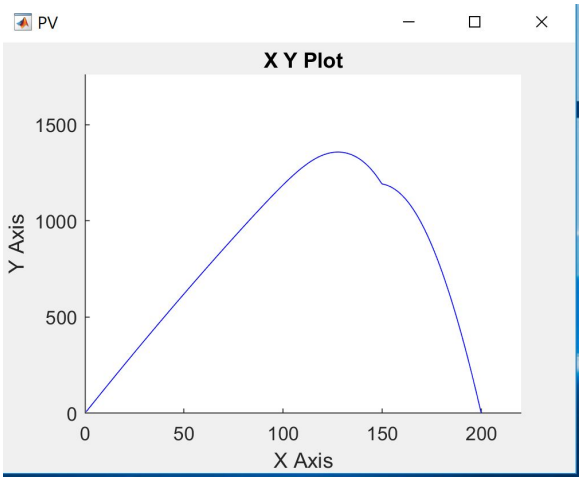

Fig 5 (c). Voltage Vs Power - One module

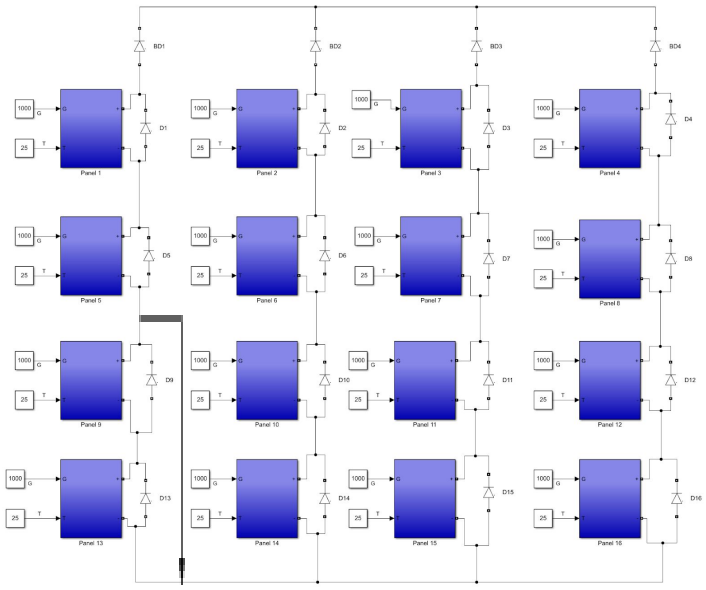

Fig 6 (a). Line-Line Fault Two module mismatch

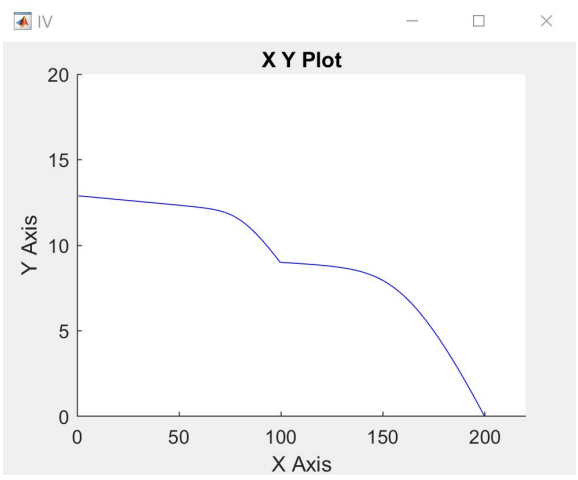

Fig 6 (b). Voltage Vs Current -Two module mismatch

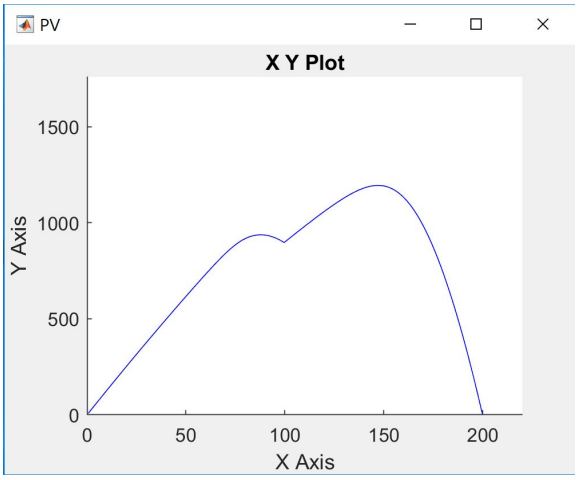

Fig 6 (c). Voltage Vs Power Two module mismatch

1.3 Machine Learning

Machine Learning (ML) is basically the process of "training" a piece of software often referred to as a model, to make useful predictions using a dataset. There are 4 general types of learning techniques namely,

- Supervised learning
- Unsupervised learning
- Semi-supervised learning
- Reinforcement learning

From the above types, in supervised learning the data has to be labelled into appropriate classes so that the model can be trained. Classification is a technique that falls under this category where it helps to classify or divide the data points in a dataset into 2 (binary classification) or more (multinomial classification) classes.

The objective of this project is to use classification algorithms to successfully identify only actual faults that occur internally. The models are trained with basic classification algorithms and ensemble techniques and a comparison of the various accuracies of the corresponding algorithms is done. The algorithms used are for this project are specified below.

1.3.1 Basic Classification:

1.3.1.1 Naive Bayes

It is based on Bayes theorem and assumes each feature is independent and has equal weightage.

Bayes Theorem is represented below by (2) which basically says that the probability of event A occurring when event B has occurred is equal to the product of the probability of event B occurring

when event A has occurred and the probability of event A occurring alone divided by the probability of B occurring alone.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \qquad (2)$$

1.3.1.2 Logistic Regression

It uses the logistic or sigmoid function and is used when the output labels are categorical in nature. The Sigmoid Function can be represented as shown below in equation (3) and Fig 7 represents the sigmoid curve.

$$sigmoid(x) = \frac{1}{1+e^{-x}} \qquad (3)$$

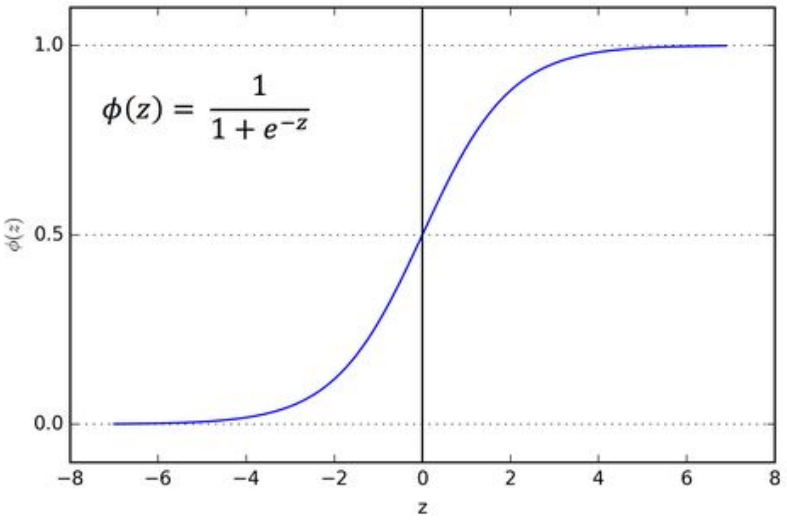

Fig 7. Sigmoid Function

Logistic regression can be used to predict probability as values lie between 0 and 1. The Logistic Regression equation based on the sigmoid function is as shown below in equation (4).

$$y = \frac{1}{1+e^{-(\beta_1 x+\beta_0)}} \quad (4)$$

where:

β_1 -> weight (slope)

β_0 -> bias (y intercept)

1.3.1.3 Decision Trees

This algorithm simply splits the data points into two sets based on a feature. Multiple splits lead to many branches that makes a decision tree which can make a decision based on the feature values at each node and classify the data point.

Decision tree builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller subsets with increase in depth of the tree. The final result is a tree with decision nodes and leaf nodes.

1.3.2 Ensemble techniques

1.3.2.1 Random forests

It takes multiple decision trees each of which chooses a subset of features all of which is aggregated into a single model. Here, each tree is built on a random sample from the original data. Also at each tree node, a subset of features are randomly selected to generate the best split.

1.3.2.2 Gradient Boosting Machine

Boosting is a methodology where weak learners are converted into strong learners. Here, each tree is fit onto a modified version of the original dataset. When it comes to Gradient Boosting, it trains models in a gradual, additive and sequential manner. Gradient boosting identifies the shortcomings of weak learners by using gradients in the loss function.

The objective is quite straightforward in that it is a classification problem and it can be fairly easily solved with quite high accuracy by using existing well established classification algorithms.

CHAPTER 2

2. Literature Survey

Fault analysis in solar photovoltaic (PV) arrays is a fundamental task to increase system reliability, efficiency, and safety. Without proper protection, faults in PV systems may damage PV modules or cables, as well as lead to dc arcing hazards and even fire risks.

[1]Y. Zhao et al., proposed that partial shading refers to non uniform irradiance in PV arrays which can be caused by dust or leaves. It could decrease power efficiency and accelerate aging of PV modules. Because of nonlinear characteristics, varying operating environment and high fault complexity of PV, many Artificial Intelligence Machine Learning algorithms are employed to automatically establish the fault diagnosis model of PV arrays.

Y. Zhao et al., proposed that a decision tree algorithm was used to detect and classify the PV array faults. Experimental data consisting of PV array current, voltage, irradiance, and operating temperature were used as attributes in the training set. The limitation of this method is that it may be difficult to obtain a training dataset that can cover all possible fault scenarios.

Chine, W et al., proposed fault detection and diagnosis methods of photovoltaic system by Artificial neural networks. In the paper the three layer feedforward neural network is used which allows the identification of the short circuit location of PV modules in one string.

Hariharan, R. et. al (2016) proposed a solution based on two variable array losses which is calculated using the difference between array power losses and instantaneous power ratios in order to detect faults and partial shading conditions.

Zhehan Yi and Amir H. Etemadi proposed a multi-resolution signal decomposition method to extract fault features and two intelligent classification methods to identify the line-to-line and ground faults. Fault detection in a PV array becomes difficult under low irradiance conditions. Line-to-line faults and ground faults may be undetected in the low irradiance situation. Artificial intelligence (AI) techniques are considered to be promising strategies to address these problems.

Chen Z C et. al proposed a method based on clustering by fast search and find density peaks (CFSFDP) which was introduced to detect and classify different faults. Moreover, optimized kernel extreme learning machine has been used to improve the effect of classification.

CHAPTER 3

3. Methodology and Results

The methodology followed for this project consists of two phases which are (a) data acquisition and labelling and (b) model training, testing and comparison of results. The first phase involves setting up the experimental setup of devices that obtain the readings from the solar PV array and is stored. The data is then labelled and other features calculated. The second phase consists of taking the now labelled data and training the various classification models with it. Finally all the results in the form of accuracy scores are calculated.

3.1 Experimental setup specifications

3.1.1 Simulink panel model specification

Isc = 3.2252 A
Voc = 50.143 V
Pmpp = 100.0053 W
Vmpp =36.977 V
Impp = 2.70 A
Rs = 2.5 ohms
Rp = 110 ohms

3.1.2 Simulink array specification

Isc = 12.895 A
Voc = 199.8714 V
Pmpp = 1592.5 W
Vmpp =146.71 V
Impp = 10.854 A

The PV system designed here has a maximum power rating (Pm) of 1592.5 W at the voltage (Vmp) of 146.71 V and current (Imp) of 10.854 A. The open-circuit voltage (V_{OCsys}) and short-circuit current (I_{SCsys}) of the PV system are 199.8714 V and 12.895 A respectively. Table 1 gives the various specifications per PV module.

Parameters	Symbol	Value
Maximum Power	Pmpp	100 W
Maximum Power Current	Impp	2.7A
Maximum Power Voltage	Vmpp	36.9V
Open-circuit Voltage	V_{OCsys}	50.14V
Short-circuit Voltage	I_{SCsys}	3.22A

Table 1: Specifications of PV module

3.2 Data Acquisition and Labelling

The solar PV system is set up in a 4 X 4 configuration. The data is obtained via a Data ACquisition (DAC) device. This is then interfaced with a computer using LabView software. The LabView model built to get the data in the correct format and the proper sampling frequency is shown in Fig 8.

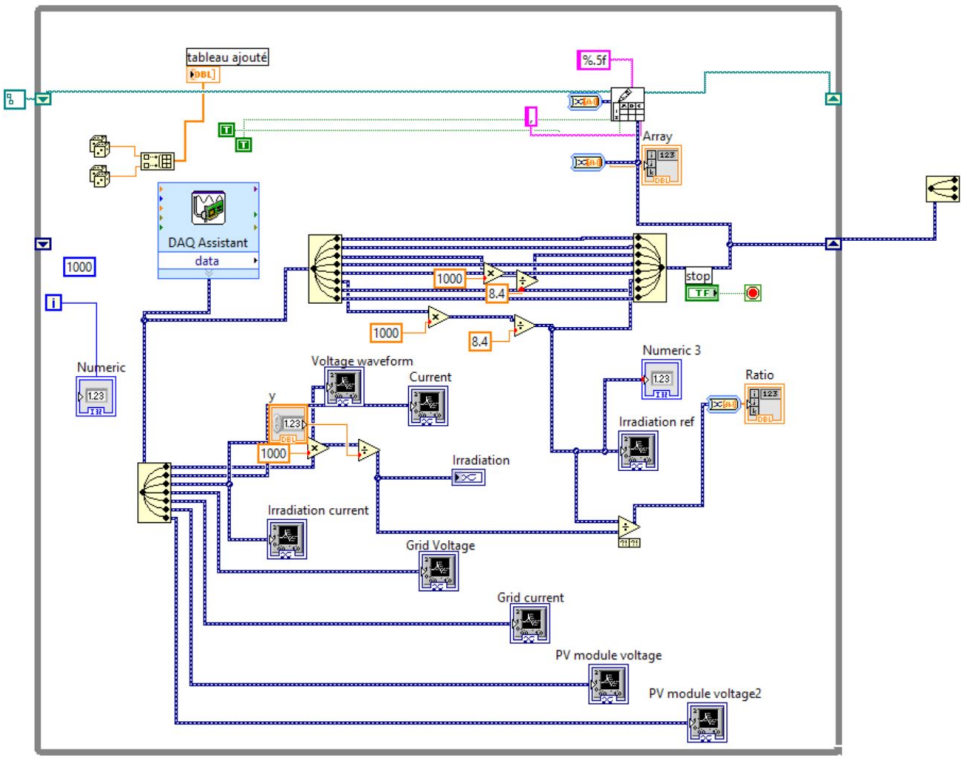

Fig 8. PV System LabView Model

3.2.1 Experimental setup

The data is continuously obtained for a period of time and stored in a CSV file. The raw data obtained are the values of Voltage (V), Current (I) and Irradiance (G). LL faults are introduced by shorting the modules with the help of a parallely connected switch. Fig 9 (a), Fig 9 (b) and Fig 9 (c) show the equipment used to get the data.

Fig 9 (a). Panel Setup

Fig 9 (b). Inverter

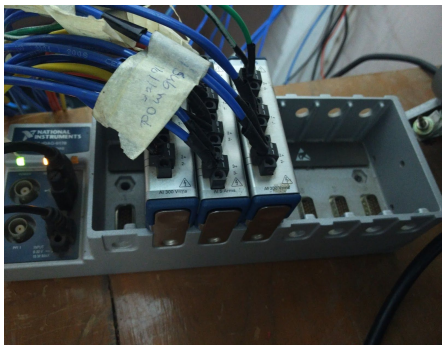

Fig 9 (c) Data Acquisition System

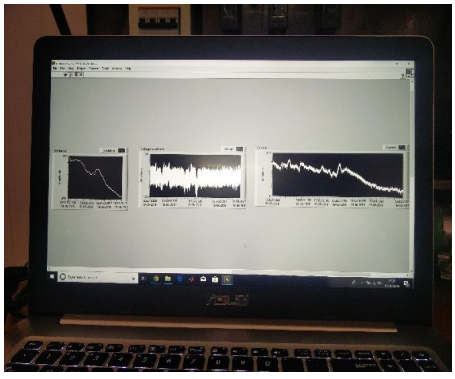

Fig 10 (a). Output V, I and G

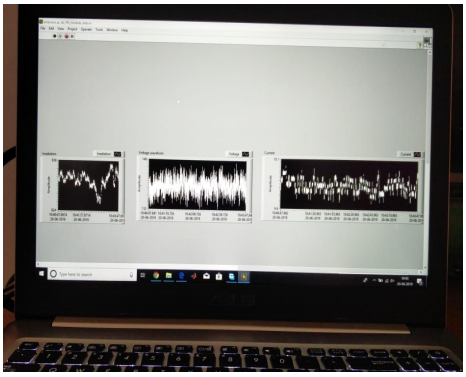

Fig 10 (b). Normal readings

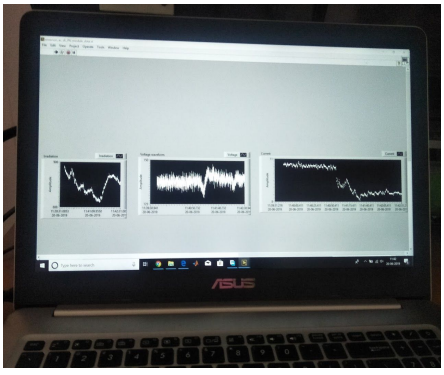

Fig 10 (c). Partial shading

The readings are represented in Fig 10 (a), Fig 10 (b) and Fig 10 (c). The partial shading on the other hand is induced by creating artificial shade on a module with the help of a plane object. Once the required data has been saved and stored, the process of labelling is done.

3.2.2 Data preprocessing and labelling

For this project, the power, P (Watts) is also calculated as a feature. The major label here is ‘fault’, where a zero denotes that there is no LL fault that has occurred and a one denotes that an LL fault has occurred. There are two more labels that have been added which give information such as the number of modules that have fault and if partial shading has occurred.

Since it is continuous data, there are no missing values in between which makes the preprocessing procedure fairly straightforward. Once the labelling is done, it is ready to be imported and worked with.

3.3 Training, testing and results

The training process involves taking the data, splitting it into train and test data sets and training on the five classification models as discussed earlier.

The data is stored in five separate CSV files, each containing data for separate conditions. They are,

1. Normal condition
2. LL1 fault in a single string
3. LL1 fault among 2 strings
4. LL2 fault in a single string
5. Partial shading of a single module

The data is imported into dataframes using the Pandas library in Python. On merging all data into one single dataframe, some preliminary data analysis is done by plotting the data. as scatter plots.

3.3.1 Data analysis

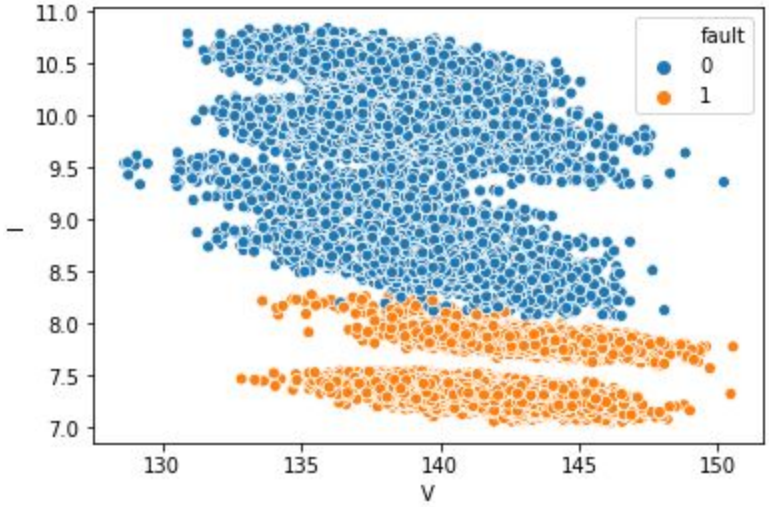

Fig 11. Current Vs Voltage plot differentiating whether a fault has occurred or not.

It is clear from Fig 11 that when a fault occurs, the value of current, I shows a significant dip. What is also interesting is that there is a clear enough separation between the fault data.

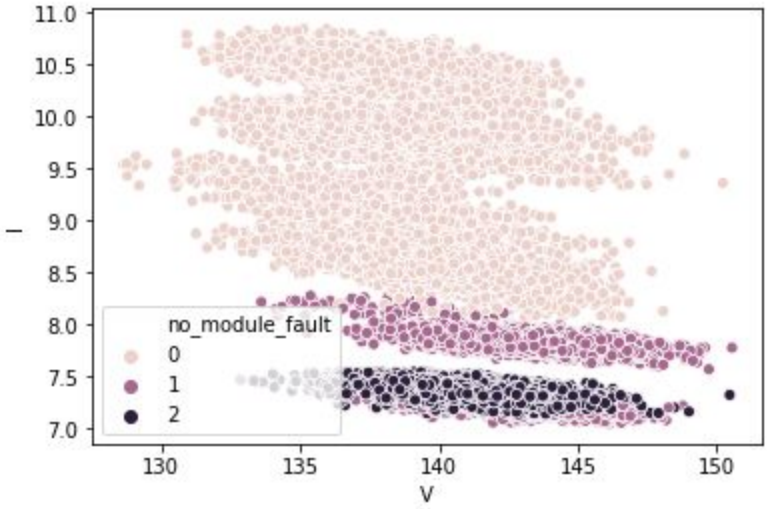

Fig 12. Current Vs Voltage plot showing clear difference between the number of modules in fault condition.

Further investigation by plotting the graph as shown in Fig 12 shows that the separation shown in Fig 11 actually is indicative of the number of modules under fault condition.

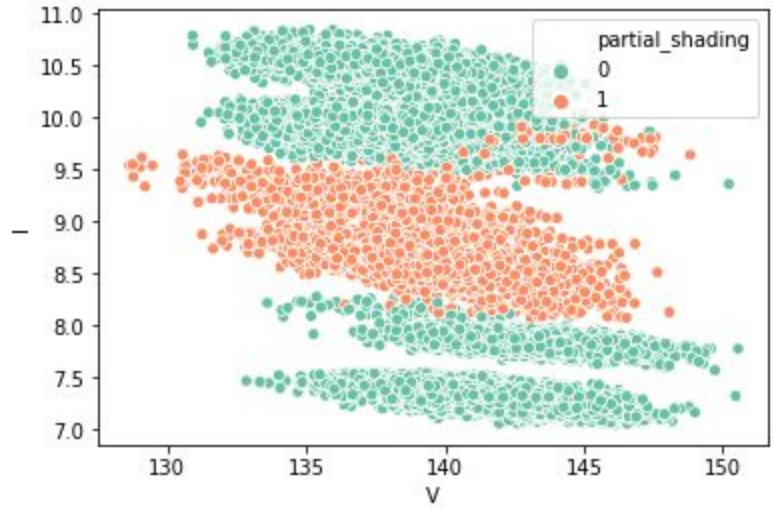

Fig 13. Current Vs Voltage plot differentiating the occurrence of partial shading

The plot in Fig 13 indicates that the drop in irradiance due to partial shading is more spread out and is not as narrow which from which we can infer that current drops due to shading tends to be more gradual than a more sudden drop in the case of internal LL faults.

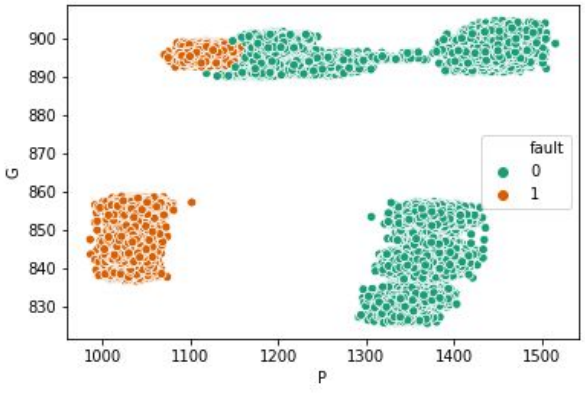

Fig 14. Irradiance Vs Power with hue as 'fault'

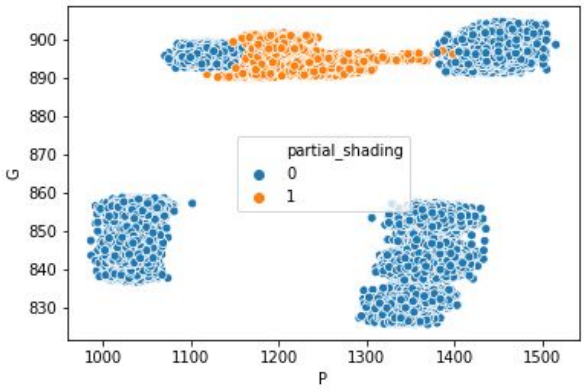

Fig 15. Irradiance Vs Power with hue as 'partial shading'

The above two plots shown in Fig 14 and Fig 15 reinforce the previous inferences that dips in current which also means dips in power (as power is the product of current and voltage) is more gradual during partial shading than when a module is under fault condition.

3.3.2 Training and testing

Training and testing is done via Stratified K-Fold cross validation with number of folds as 10. Cross validation is chosen in order to get a more generalised model and to eliminate bias as much as possible. First the data is split into X and y sets containing the features and labels respectively where y contains the main label of fault occurring or not.

Separate models are trained for each of the classifier types on the training data and tested for 10 iterations. Once trained and tested, the accuracy scores are then obtained by comparing with the actual label values and the predicted label values.

3.3.3 Results

The average of all the iterations is taken as the final accuracy score. The results in order of highest accuracy has been tabulated as shown in Fig 16.

	Classification Model	Accuracy Score (%)
2	Decision Tree	99.989455
3	Random Forest	99.987698
4	Gradient Boosting Classifier	99.973638
1	Logistic Regression	99.964851
0	Naive Bayes	99.833040

Fig 16. Accuracy score comparison for fault detection

Thus it shows that the best accuracy is gotten from using the decision tree and then the two ensemble classifiers. The accuracy for GBM could be increased marginally by tweaking the hyper

parameter of n estimators to 109. Naive Bayes fairs the worst among these probably because it helps to consider dependency among the features such as power or current and irradiance.

Hence from the analysis and comparison, we see a very high accuracy overall with Decision Tree algorithm faring the best and of course Random Forest which is an ensemble of decision trees coming close behind.

	Classification Model	Accuracy Score (%)
3	Random Forest	99.982425
2	Decision Tree	99.966608
4	Gradient Boosting Classifier	99.966608
0	Naive Bayes	99.040422
1	Logistic Regression	91.905097

Fig 17. Accuracy score comparison for partial shading

	Classification Model	Accuracy Score (%)
4	Gradient Boosting Classifier	99.732865
3	Random Forest	99.708260
2	Decision Tree	99.704745
1	Logistic Regression	94.063269
0	Naive Bayes	94.015817

Fig 18. Accuracy score comparison for no. of modules fault

Now the same data is used to train for classifying partial shading and also for the number of modules having faults whose results are shown in Fig 17 and Fig 18 respectively.

Here, interestingly for partial shading, Logistic Regression is significantly lower than the rest while Naive Bayes even though is much better still comes significantly lower when compared to the other three algorithms. This indicates that Decision Tree and subsequently Random Forest which is more generalised fairs the best most of the time when it comes to binary classification. However, while testing for number of modules under fault condition, GBM has come out ahead indicating that it might have an edge over the others when it comes to multiclass classification.

Future work can include collecting more data over many day-cycles and seasons in order to generalise it as much as possible. Also, if data of actual faults occurring is taken instead of artificially induced faults, the amount of data points for fault conditions will be low and hence other metrics such as precision, recall and F1 scores should be taken into account to test for the authenticity of the models' predictive capabilities.

Appendix

Sample Code

```
# Importing various required libraries
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
from sklearn.metrics import accuracy_score
# Only normal data
data_normal = pd.read_csv("Final_Solar_Data/normal.csv")
# LL1
data_LL1 = pd.read_csv("Final_Solar_Data/LL1.csv")
# LL2
data_LL2 = pd.read_csv("Final_Solar_Data/LL2.csv")
# Str1_3-Str2_2-LL1
data_LL1_str = pd.read_csv("Final_Solar_Data/Str1_3-Str2_2-LL1.csv")
# Partial Shading
data_partial = pd.read_csv("Final_Solar_Data/Partial_shading.csv")
# Appending all data together
total = pd.concat([data_normal, data_LL1, data_LL2, data_LL1_str, data_partial], ignore_index=True)
# Splitting features and labels for training and testing
total_X = total.drop(['no_module_fault', 'fault', 'partial_shading'], axis=1)
total_y = total['fault']
total_y_partial = total['partial_shading']
total_y_modules = total['no_module_fault']
# Data Analysis by plotting scatter plots using seaborn
```

```
sns.scatterplot(x="V", y="I", hue="fault", data=total) # Fig 11

sns.scatterplot(x="V", y="I", hue="no_module_fault", data=total) # Fig 12

sns.scatterplot(x="V", y="I", hue="partial_shading", data=total, palette="Set2") # Fig 13

sns.scatterplot(x="P", y="G", hue="partial_shading", data=total) # Fig 14

sns.scatterplot(x="P", y="G", hue="fault", data=total, palette="Dark2") # Fig 15

total_shuffled = total.sample(frac=1) # shuffling the data

# Cross validation

from sklearn.model_selection import StratifiedKFold

from sklearn.model_selection import cross_val_score

SSS = StratifiedKFold(n_splits=10)

# Naive Bayes

model_GNB_crossval = GaussianNB()

predict_GNB_crossval = cross_val_score(model_GNB_crossval, total_X, total_y, cv=SSS.split(total_X,
total_y))

avg_crossval_score_GNB = predict_GNB_crossval.mean()

print("Cross validated accuracies:", predict_GNB_crossval)

print("Average cross validated accuracy:", avg_crossval_score_GNB*100, "%")
```

Out:

Cross validated accuracies: [1. 1. 1. 1. 1. 0.9970123 0.99367311 0.99613357 1. 0.99648506]

Average cross validated accuracy: 99.83304042179262 %

```
# Naive Bayes Partial

model_GNB_crossval_partial = GaussianNB()

predict_GNB_crossval_partial = cross_val_score(model_GNB_crossval_partial, total_X, total_y_partial,
cv=SSS.split(total_X, total_y_partial))

avg_crossval_score_GNB_partial = predict_GNB_crossval_partial.mean()

print("Cross validated accuracies:", predict_GNB_crossval_partial)

print("Average cross validated accuracy:", avg_crossval_score_GNB_partial*100, "%")
```

Out:

Cross validated accuracies: [0.99191564 0.99050967 0.99173989 0.99068541 0.98963093 0.98927944

0.98769772 0.99103691 0.99209139 0.98945518]

Average cross validated accuracy: 99.04042179261863 %

```
# Naive Bayes Modules

model_GNB_crossval_modules = GaussianNB()

predict_GNB_crossval_modules = cross_val_score(model_GNB_crossval_modules, total_X, total_y_modules, cv=SSS.split(total_X, total_y_modules))

avg_crossval_score_GNB_modules = predict_GNB_crossval_modules.mean()

print("Cross validated accuracies:", predict_GNB_crossval_modules)

print("Average cross validated accuracy:", avg_crossval_score_GNB_modules*100, "%")
```

Out:

Cross validated accuracies: [0.86502636 0.86520211 0.90615114 0.99806678 0.99929701 0.98927944

0.94534271 0.94393673 0.90949033 0.9797891]

Average cross validated accuracy: 94.0158172231986 %

```
# Logistic Regression

model_LogReg_crossval = LogisticRegression()

predict_LogReg_crossval = cross_val_score(model_LogReg_crossval, total_X, total_y, cv=SSS.split(total_X, total_y))

avg_crossval_score_LogReg = predict_LogReg_crossval.mean()

print("Cross validated accuracies:", predict_LogReg_crossval)

print("Average cross validated accuracy:", avg_crossval_score_LogReg*100, "%")
```

Out:

Cross validated accuracies: [1. 1. 1. 1. 1. 0.99982425

0.99789104 0.99912127 1. 0.99964851]

Average cross validated accuracy: 99.96485061511426 %

```
# Logistic Regression Partial

model_LogReg_crossval_partial = LogisticRegression()
```

```
predict_LogReg_crossval_partial = cross_val_score(model_LogReg_crossval_partial, total_X,
total_y_partial, cv=SSS.split(total_X, total_y_partial))

avg_crossval_score_LogReg_partial = predict_LogReg_crossval_partial.mean()

print("Cross validated accuracies:", predict_LogReg_crossval_partial)

print("Average cross validated accuracy:", avg_crossval_score_LogReg_partial*100, "%")
```

Out:

Cross validated accuracies: [0.94780316 0.94991213 0.9483304 0.93813708 0.93936731
0.88347979

0.81546573 0.92073814 0.94130053 0.9059754]

Average cross validated accuracy: 91.90509666080844 %

```
# Logistic Regression Modules

model_LogReg_crossval_modules = LogisticRegression()

predict_LogReg_crossval_modules = cross_val_score(model_LogReg_crossval_modules, total_X,
total_y_modules, cv=SSS.split(total_X, total_y_modules))

avg_crossval_score_LogReg_modules = predict_LogReg_crossval_modules.mean()

print("Cross validated accuracies:", predict_LogReg_crossval_modules)

print("Average cross validated accuracy:", avg_crossval_score_LogReg_modules*100, "%")
```

Out:

Cross validated accuracies: [0.85377856 0.85711775 0.89490334 0.98435852 0.98681898
0.98224956

0.98365554 0.98101933 0.90105448 0.98137083]

Average cross validated accuracy: 94.06326889279437 %

```
# Decision Trees

model_DT_crossval = DecisionTreeClassifier()

predict_DT_crossval = cross_val_score(model_DT_crossval, total_X, total_y, cv=SSS.split(total_X, total_y))

avg_crossval_score_DT = predict_DT_crossval.mean()

print("Cross validated accuracies:", predict_DT_crossval)

print("Average cross validated accuracy:", avg_crossval_score_DT*100, "%")
```

Out:

Cross validated accuracies: [1. 1. 1. 1. 1. 1.

0.99912127 1. 1. 0.99982425]

Average cross validated accuracy: 99.98945518453426 %

```
# Decision Trees Partial

model_DT_crossval_partial = DecisionTreeClassifier()

predict_DT_crossval_partial = cross_val_score(model_DT_crossval_partial, total_X, total_y_partial,
cv=SSS.split(total_X, total_y_partial))

avg_crossval_score_DT_partial = predict_DT_crossval_partial.mean()

print("Cross validated accuracies:", predict_DT_crossval_partial)

print("Average cross validated accuracy:", avg_crossval_score_DT_partial*100, "%")
```

Out:

Cross validated accuracies: [0.99982425 0.99947276 1. 0.99964851 0.99947276 0.99929701

0.99982425 1. 0.99912127 1.]

Average cross validated accuracy: 99.96660808435853 %

```
# Decision Trees Modules

model_DT_crossval_modules = DecisionTreeClassifier()

predict_DT_crossval_modules = cross_val_score(model_DT_crossval_modules, total_X, total_y_modules,
cv=SSS.split(total_X, total_y_modules))

avg_crossval_score_DT_modules = predict_DT_crossval_modules.mean()

print("Cross validated accuracies:", predict_DT_crossval_modules)

print("Average cross validated accuracy:", avg_crossval_score_DT_modules*100, "%")
```

Out:

Cross validated accuracies: [0.99332162 0.99613357 0.99648506 0.99771529 0.99894552
0.99824253

0.99841828 0.99753954 0.99543058 0.99824253]

Average cross validated accuracy: 99.70474516695958 %

```
# Random Forest

model_RF_crossval = RandomForestClassifier()
```

```
predict_RF_crossval = cross_val_score(model_RF_crossval, total_X, total_y, cv=SSS.split(total_X, total_y))

avg_crossval_score_RF = predict_RF_crossval.mean()

print("Cross validated accuracies:", predict_RF_crossval)

print("Average cross validated accuracy:", avg_crossval_score_RF*100, "%")
```

Out:

```
Cross validated accuracies: [1.        1.        1.        1.        1.        1.
 0.99912127 0.99964851 1.        1.        ]
Average cross validated accuracy: 99.98769771528998 %
```

```
# Random Forest partial

model_RF_crossval_partial = RandomForestClassifier()

predict_RF_crossval_partial = cross_val_score(model_RF_crossval_partial, total_X, total_y_partial,
cv=SSS.split(total_X, total_y_partial))

avg_crossval_score_RF_partial = predict_RF_crossval_partial.mean()

print("Cross validated accuracies:", predict_RF_crossval_partial)

print("Average cross validated accuracy:", avg_crossval_score_RF_partial*100, "%")
```

Out:

```
Cross validated accuracies: [0.99982425 0.99964851 0.99982425 0.99982425 0.99964851
0.99982425
 0.99964851 1.        1.        1.        ]
Average cross validated accuracy: 99.98242530755711 %
```

```
# Random Forest modules

model_RF_crossval_modules = RandomForestClassifier()

predict_RF_crossval_modules = cross_val_score(model_RF_crossval_modules, total_X, total_y_modules,
cv=SSS.split(total_X, total_y_modules))

avg_crossval_score_RF_modules = predict_RF_crossval_modules.mean()

print("Cross validated accuracies:", predict_RF_crossval_modules)

print("Average cross validated accuracy:", avg_crossval_score_RF_modules*100, "%")
```

Out:

```
Cross validated accuracies: [0.99332162 0.99560633 0.99666081 0.99841828 0.99912127
0.99824253
```

```
 0.99841828 0.9970123  0.99595782 0.99806678]
Average cross validated accuracy: 99.70826010544815 %
```

```
# Gradient Boosting Machine
model_GBM_crossval = GradientBoostingClassifier()
predict_GBM_crossval = cross_val_score(model_GBM_crossval, total_X, total_y, cv=SSS.split(total_X, total_y))
avg_crossval_score_GBM = predict_GBM_crossval.mean()
print("Cross validated accuracies:", predict_GBM_crossval)
print("Average cross validated accuracy:", avg_crossval_score_GBM*100, "%")
```

Out:

```
Cross validated accuracies: [1.        1.        1.        1.        1.        1.
 0.99841828 0.99894552 1.        0.99947276]
Average cross validated accuracy: 99.96836555360282 %
```

```
# Gradient Boosting Machine with n_estimators=109 (hyper-parameter tweaking)
model_GBM_crossval = GradientBoostingClassifier(n_estimators=109)
predict_GBM_crossval = cross_val_score(model_GBM_crossval, total_X, total_y, cv=SSS.split(total_X, total_y))
avg_crossval_score_GBM = predict_GBM_crossval.mean()
print("Cross validated accuracies:", predict_GBM_crossval)
print("Average cross validated accuracy:", avg_crossval_score_GBM*100, "%")
```

Out:

```
Cross validated accuracies: [1.        1.        1.        1.        1.        1.
 0.99859402 0.99912127 1.        0.99964851]
Average cross validated accuracy: 99.97363796133568 %
```

```
# Gradient Boosting Machine partial
model_GBM_crossval_partial = GradientBoostingClassifier(n_estimators=109)
```

```
predict_GBM_crossval_partial = cross_val_score(model_GBM_crossval_partial, total_X, total_y_partial,
cv=SSS.split(total_X, total_y_partial))

avg_crossval_score_GBM_partial = predict_GBM_crossval_partial.mean()

print("Cross validated accuracies:", predict_GBM_crossval_partial)

print("Average cross validated accuracy:", avg_crossval_score_GBM_partial*100, "%")
```

Out:

```
        Cross validated accuracies: [1.        0.99929701 0.99982425 0.99964851 0.99947276 0.99929701
         0.99982425 1.        0.99964851 0.99964851]
        Average cross validated accuracy: 99.96660808435853 %
```

```
# Gradient Boosting Machine modules

model_GBM_crossval_modules = GradientBoostingClassifier(n_estimators=109)

predict_GBM_crossval_modules = cross_val_score(model_GBM_crossval_modules, total_X,
total_y_modules, cv=SSS.split(total_X, total_y_modules))

avg_crossval_score_GBM_modules = predict_GBM_crossval_modules.mean()

print("Cross validated accuracies:", predict_GBM_crossval_modules)

print("Average cross validated accuracy:", avg_crossval_score_GBM_modules*100, "%")
```

Out:

```
        Cross validated accuracies: [0.99297012 0.99648506 0.99613357 0.99841828 0.99859402
0.99824253
         0.99912127 0.99912127 0.99560633 0.99859402]
        Average cross validated accuracy: 99.7328646748682 %
```

```
# Comparing the results

# Cross-validated scores for fault detection

AccList_Crossval = [['Naive Bayes', avg_crossval_score_GNB*100],
        ['Logistic Regression', avg_crossval_score_LogReg*100],
        ['Decision Tree', avg_crossval_score_DT*100],
        ['Random Forest', avg_crossval_score_RF*100],
        ['Gradient Boosting Classifier', avg_crossval_score_GBM*100]
        ]
```

```
acc_results = pd.DataFrame(AccList_Crossval, columns = ['Classification Model', 'Accuracy Score (%)'])

acc_results.sort_values(by=['Accuracy Score (%)'], inplace=True, ascending=False) # Sorting records in order of highest accuracy

acc_results.head() # Fig 16

# Cross-validated scores for partial shading predictions

AccList_Crossval = [['Naive Bayes', avg_crossval_score_GNB_partial*100],

        ['Logistic Regression', avg_crossval_score_LogReg_partial*100],

        ['Decision Tree', avg_crossval_score_DT_partial*100],

        ['Random Forest', avg_crossval_score_RF_partial*100],

        ['Gradient Boosting Classifier', avg_crossval_score_GBM_partial*100]

        ]

acc_results = pd.DataFrame(AccList_Crossval, columns = ['Classification Model', 'Accuracy Score (%)'])

acc_results.sort_values(by=['Accuracy Score (%)'], inplace=True, ascending=False) # Sorting records in order of highest accuracy

acc_results.head() # Fig 17

# Cross-validated scores for number of modules in fault condition

AccList_Crossval = [['Naive Bayes', avg_crossval_score_GNB_modules*100],

        ['Logistic Regression', avg_crossval_score_LogReg_modules*100],

        ['Decision Tree', avg_crossval_score_DT_modules*100],

        ['Random Forest', avg_crossval_score_RF_modules*100],

        ['Gradient Boosting Classifier', avg_crossval_score_GBM_modules*100]

        ]

acc_results = pd.DataFrame(AccList_Crossval, columns = ['Classification Model', 'Accuracy Score (%)'])

acc_results.sort_values(by=['Accuracy Score (%)'], inplace=True, ascending=False) # Sorting records in order of highest accuracy

acc_results.head() # Fig 18
```