Gold Tree Solar Farm - Machine Learning to Predict Solar Power Generation

Jonathon Scott

supervised by
Dr. Phillip Nico, Computer Science Department
Dr. Jacques Belanger, Mechanical Engineering Department

June 6, 2019
## Contents

1 Introduction .................................................. 3
   1.1 Problem Statement ......................................... 3
   1.2 Scope of Work .............................................. 3

2 Background ..................................................... 3
   2.1 Photovoltaic Solar Power .................................. 3
   2.2 Gold Tree Solar Farm (GTSF) ............................... 5
   2.3 Green Power Monitor (GPM) ............................... 6
   2.4 PlantPredict ................................................. 7
   2.5 Images In Computation .................................... 7
   2.6 Machine Learning .......................................... 9

3 Implementation ................................................. 11
   3.1 Green Power Monitor API .................................. 11
      3.1.1 Exploring the Green Power Monitor API ............. 11
      3.1.2 Authentication ......................................... 11
      3.1.3 Get All Plants ......................................... 12
      3.1.4 Get All Data Sources .................................. 12
      3.1.5 Get Data List .......................................... 14
   3.2 Gold Tree Software Development Kit (SDK) ............... 15
      3.2.1 Why build an SDK? ..................................... 15
      3.2.2 How to use the SDK .................................... 16
      3.2.3 Publishing a Python Package .......................... 16
   3.3 Data Curation ................................................ 17
      3.3.1 Exploration ............................................. 17
      3.3.2 Interval Selection ...................................... 20
      3.3.3 Cleaning ................................................ 21
   3.4 Sky Image Processing ....................................... 21
      3.4.1 Red Blue Ratio ........................................... 22
   3.5 Weatherbit.io ................................................. 23
   3.6 PlantPredict API/SDK ...................................... 24
      3.6.1 Authentication .......................................... 24
      3.6.2 Uploading Weather Data ............................... 24
      3.6.3 Running a Simulation ................................... 25
   3.7 Machine Learning ............................................ 25
1. Introduction

1.1. Problem Statement

Solar energy causes a strain on the electrical grid because of the uncontrollable nature of the factors that affect power generation. Utilities are often required to balance solar generation facilities to meet consumer demand, which often includes the costly process of activating/deactivating a fossil fuel facility. Therefore, there is considerable interest in increasing the accuracy and the granularity of solar power generation predictions in order to reduce the cost of grid management. This project aims to evaluate how sky imaging technology may contribute to the accuracy of those predictions.

1.2. Scope of Work

The scope of this project includes data collection and cleaning, creation of predictive models, and the use of those models to predict power generation. The raw data is to be collected from Cal Poly’s Gold Tree solar farm. The data will be composed of power generation measurements and meteorological measurements such as temperature, wind speed, humidity, solar irradiance. Furthermore, image data from the onsite sky camera will be used. This raw data will then need to be cleaned of illogical or missing data. It will then be used to train machine learning models in order to predict power generation. These machine learning models will be compared with the PlantPredict simulation created by Isabella Kueschle in her senior project during Winter 2018. Through this process, PlantPredict will be evaluated on its suitability for short-term forecasting. Machine learning models will investigated as be forecasting alternatives.

2. Background

2.1. Photovoltaic Solar Power

Photovoltaic solar systems convert sunlight directly into electricity for human consumption. The efficiency of these systems are directly dependent on the amount of sunlight available to them. Irradiance is the instantaneous measurement of solar power hitting an surface, while insolation is a measurement of the sum of the solar power in on a surface over a period of time. There exist three different types of irradiance measurements: GHI, DNI, and DHI. GHI (global horizontal irradiance) is a sum of all solar radiation on a horizontal surface. DNI (direct normal irradiance) is a measurement of the
irradiance attributable to light energy on a surface kept perpendicular as sun travels across the sky.

Figure 1: Illustration of Direct Normal Irradiance

DHI (diffuse horizontal irradiance) is a measurement of the irradiance attributable to light energy that has diffused throughout the atmosphere. DHI exists because light refracts and scatters upon entering the Earth’s atmosphere.

Figure 2: Illustration of Diffuse Horizontal Irradiance

The angle of a solar panel plays an important role in the efficiency of that panel. Some systems are stationary and remain at the same angle throughout a solar day and miss out generating additional energy. In order to maintain the most efficient angle with the sun, some systems implement single axis tracking in which the panel will rotate along a single axis tracking the sun
across the sky (typically in an east to west fashion). However, due to the tilt of the Earth, the sun’s path across the sky is not constant - it is higher in the sky during the summer months and lower during the winter months. This “height” in the sky is known as the azimuth. Dual axis tracking permits a panel to align with the sun throughout the day and throughout the seasonal shifting of the sun’s path.

2.2. Gold Tree Solar Farm (GTSF)

The California Polytechnic State University of San Luis Obispo (Cal Poly) has set a goal of carbon neutrality by the year 2050. In their efforts towards achieving this goal, Cal Poly completed the 4.5 MW Gold Tree Solar Farm in May of 2018. The Gold Tree facility was constructed under a power purchase agreement (PPA) with Duke Energy Renewables. A PPA allows Cal Poly to purchase the energy generated by the facility at a discounted rate and avoid the upfront and continual costs of construction and maintenance. The solar farm is intended to serve multiple functions for Cal Poly. Firstly, it will meet 25% of Cal Poly’s energy needs. Secondly, it will serve as a valuable resource for academic study. The performance data from the solar farm is collected and made available for Cal Poly’s ongoing efforts in solar research [1].

The Gold Tree Solar Farm consists of 16,000 solar panels and utilizes single-axis tracking technology. The facility collects energy production data down to the inverter level and has multiple weather sensors on site. Furthermore, there are several cameras at the facility, one of which is a sky camera.

Figure 3: An Image of the Gold Tree Solar Farm
that captures an image of the sky every five minutes. The site is located at 35.32 latitude and -120.69 longitude, which is roughly two miles northwest of the Cal Poly Campus.

Figure 4: Google Maps Location of Gold Tree Solar Farm

2.3. Green Power Monitor (GPM)

Green Power Monitor is the system of record that has been chosen to be the data steward for the Gold Tree Solar Farm. The meteorological and power production data collected by the Gold Tree Solar Farm is available to students through a web application with a graphical user interface and an API with Swagger documentation. The granularity of the data available to users is limited to a five minutes when querying data through the "Customized Query" functionality of the web application. From the API, however, data can be received at an one minute granularity [2].
2.4. PlantPredict

PlantPredict is a cloud-based solar system modeling tool developed by First Solar [3]. PlantPredict provides an online web interface and an API where users can piece together a proposed solar plant from individual components. Once all of the smaller modules of a solar plant have been composed to represent the full system, the user may select historical weather data for the plant’s proposed location and fire off a simulation.

This project did not modify Isabella’s original model of the Gold Tree Solar Farm, but interacted with it incorporating weather data collected from the sensors on the Gold Tree facility in the simulation. The output of the simulation was then compared to the known values for power generation provided by the GPM API.

2.5. Images In Computation

In computation, a black and white image, also called a gray scale image, is often represented as a 2 dimensional matrix. One dimension is for the width and one dimension is for the height of the image. A 8x8 image would contain 64 points, while a 256x256 would contain 65,536 points and so on. The values contained within each point in the matrix, the pixels, contain values between 0-255. A pixel with a value of 0 is black, while a pixel with a value of 255 is white. A pixel with a value somewhere between the two is a shade of gray. The alignments of the pixels with their varying intensities combine to reveal the contours, lines, and shading of the image. The entire matrix of a gray scale image is considered to be one channel of information [4].
A colored image has three channels of information. Under the Red, Green, and Blue (RGB) coloring scheme, each color has its own matrix of the same dimensions as the entire image. Each channel represents the intensity of the color of light for the image. If each channel of an RGB image was rendered separately, it would appear similar to an equivalent gray scale image. However, differences between the different channels will be noticeable. This is because the colored channels represent different information about the same scene. Blue skies would appear brighter in the blue channel and duller in the red and green channels. Green foliage would appear brighter in the green channel and duller in the red and blue channels.

In the following series of images, observe how the brightness of each of the pins change as each color channel is displayed. Notice that the red, green, and blue pins each become brighter in the image that displays their channel’s color. Furthermore, notice that the yellow pin in the middle is bright in both the red and green channels. This is because the red and green channels combine together to create the yellow color.
2.6. Machine Learning

Machine learning is the process of building a predictive model from observations in data. An observation is defined to represent an entity that is being measured. The measurements that correspond to such an entity are

<table>
<thead>
<tr>
<th>Image</th>
<th>Channel</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.jpg" alt="Original" /></td>
<td>Original</td>
</tr>
<tr>
<td><img src="image2.jpg" alt="Red" /></td>
<td>Red</td>
</tr>
<tr>
<td><img src="image3.jpg" alt="Green" /></td>
<td>Green</td>
</tr>
<tr>
<td><img src="image4.jpg" alt="Blue" /></td>
<td>Blue</td>
</tr>
</tbody>
</table>

Table 1: Illustration of Image Channels[5]
represented by variables. A machine learning model requires two types of variables: predictors and response. In order to construct a model, data for both the predictors and response variables are required. Then, a model is fit to the existing data (called the training data) so that when additional data (called future data) becomes available the model will be able to predict the response variables for the future data. There are two major categories of machine learning models. The first, regression models, are used to predict numerical values. The other, classification models, are used to predict whether an observation belongs to a certain class. The simplest regression algorithm of machine learning would be a linear regression. Another regression algorithm, called K-Nearest Neighbors, examines a given observation and determines the other observations of variable K that are closest to the given observation. These other observations, the neighbors, are determined by computing the distance between them based on the selected set of predictor variables. Then, the response variable for the given observation is defined to be the mean of the response variables of the neighbors. For this model, if the given observation was part of the training data, the output generated by K-Nearest Neighbors can be compared with the known value of the given observation to calculate the training error.

There exists a metric called test error in which a machine learning model predicts the response variables on data that it is not trained on. An estimate for test error can be determined for a model through the use of cross-validation. Cross-validation is the practice of splitting a data set into multiple subsets and training machine learning models on the subsets. The models are then used to predict the response variables in the subsets of data that they were not trained on. In this way, we are able to generate an estimate of how good a machine learning model will be at predicting future data [6].

Machine learning models are capable of learning patterns within the data to such an extent that the patterns they learn do not generalize to new data. A model that cannot generalize does not perform well on data that it has not been trained on. This is a big deal because models are ultimately trained for the purpose of predicting new data. When a model memorizes non-general patterns in the training data it is called over fitting. On the other hand, a model that fails to learn any patterns is said to under fit the data because it does not perform well even on the training data. The tuning of machine learning models is largely a battle between finding the balance between overfitting and underfitting [6][7].
3. Implementation

3.1. Green Power Monitor API

3.1.1. Exploring the Green Power Monitor API

The documentation for using the GPM API for this project was extremely sparse. The user manual that I received provided enough information to send an authentication request to the GPM servers and get back an access token. For an small API, Swagger documentation typically suffices. However, an API with 480 exposed endpoints like the GPM API is unwieldy without additional documentation. In order to find the few endpoints that I needed for this project, I had to try endpoints until I got the data I was looking for. Then, just to be certain that the data represented the measurements that I thought they represented, I validated it against the data available from the web interface. In order to prevent anyone from repeating this exact process of exploration, I have listed endpoints of interest below.

3.1.2. Authentication

All requests to the GPM API require a valid access token. The access token must be sent in the header of all subsequent requests. A token may be acquired by sending the username and password of the API credentials to the endpoint below.

Figure 6: GPM API Documentation for Authentication
3.1.3. *Get All Plants*

This request returns all of the solar facilities visible to the authenticated user which is important for determining the facility/plant id of the GTSF. The facility id is static, therefore this request will only need to be made once. For this project, it was discovered that the GTSF facility id is 42.

![Figure 7: GPM API Documentation for Get All Plants](image)

3.1.4. *Get All Data Sources*

This request returns all of the data sources that are configured for the facility. This is large amount of data so I recommend saving the body of the response into a file. The API response is JSON but I converted it to CSV via an online tool. The file should then be explored through in some sort of spreadsheet software. I was able to discover the datasource id’s for power production and weather metrics from this file.
Figure 8: GPM API Documentation for Get All Data Sources
Table 2: Significant Gold Tree Data Source Ids

<table>
<thead>
<tr>
<th>DataSourceIdentifier</th>
<th>Id</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>29372</td>
<td>1</td>
<td>Power</td>
<td>TotalPower</td>
</tr>
<tr>
<td>29373</td>
<td>2</td>
<td>Energy</td>
<td>TodayEnergy</td>
</tr>
<tr>
<td>29374</td>
<td>3</td>
<td>PR</td>
<td>PerformanceRatio</td>
</tr>
<tr>
<td>29378</td>
<td>4</td>
<td>Availability</td>
<td>Availability</td>
</tr>
<tr>
<td>27988</td>
<td>5</td>
<td>Budgeted Energy</td>
<td>EstimatedProduction</td>
</tr>
<tr>
<td>20000</td>
<td>6</td>
<td>Budgeted PR</td>
<td>EstimatedPerformanceRatio</td>
</tr>
<tr>
<td>27997</td>
<td>7</td>
<td>Budgeted Availability</td>
<td>None</td>
</tr>
<tr>
<td>27999</td>
<td>8</td>
<td>Budgeted Insolation</td>
<td>None</td>
</tr>
<tr>
<td>29300</td>
<td>9</td>
<td>Specific Power</td>
<td>None</td>
</tr>
<tr>
<td>29381</td>
<td>10</td>
<td>Specific Energy</td>
<td>None</td>
</tr>
<tr>
<td>27991</td>
<td>11</td>
<td>Specific Irradiance</td>
<td>None</td>
</tr>
<tr>
<td>27985</td>
<td>12</td>
<td>Irradiance</td>
<td>Irradiance</td>
</tr>
<tr>
<td>27988</td>
<td>13</td>
<td>Insolation</td>
<td>Irradiation</td>
</tr>
<tr>
<td>27987</td>
<td>14</td>
<td>Cell Temperature</td>
<td>WorkingTemperature</td>
</tr>
<tr>
<td>27984</td>
<td>15</td>
<td>Ambient Temperature</td>
<td>AmbientTemperature</td>
</tr>
<tr>
<td>27993</td>
<td>16</td>
<td>Wind Speed</td>
<td>WindSpeed</td>
</tr>
<tr>
<td>27992</td>
<td>17</td>
<td>Wind Direction</td>
<td>WindDirection</td>
</tr>
<tr>
<td>29379</td>
<td>18</td>
<td>Availability with Exceptions</td>
<td>None</td>
</tr>
<tr>
<td>27982</td>
<td>27</td>
<td>Precipitation</td>
<td>Precipitation</td>
</tr>
<tr>
<td>27990</td>
<td>28</td>
<td>Relative Humidity</td>
<td>RelativeHumidity</td>
</tr>
<tr>
<td>27983</td>
<td>31</td>
<td>Air Pressure</td>
<td>AirPressure</td>
</tr>
</tbody>
</table>

From the table above we can see that data source id 29373 identifies the amount of energy produced by the plant and that id 27984 identifies the ambient temperature recorded at the plant.

3.1.5. Get Data List

This request uses query parameters in the request url to get specific data from the GPM API. This is a complex end point with a variety of configuration options, so I would advise the reader to read GPM's own documentation for this method. However, I will provide a simple explanation for getting started. It is important to note that the startDate and endDate are expected to be numeric Unix-epoch time stamps. The aggregationType field provides a wide array of options that I chose not to explore and stayed with the value of 0. Grouping determines the granularity of the returned data. I used the following values for grouping: raw, tenminute, and hour.
3.2. Gold Tree Software Development Kit (SDK)

3.2.1. Why build an SDK?

When I first encountered the Green Power Monitor API, I felt overwhelmed by the number of endpoints available to query. After a significant amount of tinkering and exploring the GPM API, I eventually found the small subset of endpoints of interest for the project. I found the process of manually referencing data streams by looking them up with their unique identifier in a large excel sheet of other identifiers to be error prone and inefficient. In order to simplify the process of obtaining data from the GPM API, I decided to build a python package that would simplify the process of obtaining data from the Gold Tree Solar Farm.
3.2.2. How to use the SDK

A user of the SDK must have valid credentials for the GPM API. I was able to obtain valid credentials by reaching out to Cal Poly’s Sustainability Department who put me in contact with a representative from REC Solar who provided me with credentials. Once credentials have been acquired, the user will need to install the SDK using pip. After the package has installed, the user will need to modify the settings.py configuration to contain the username and password of the credentials. It is important to note that the password will be stored locally within this file in clear text and should be considered a serious security concern for any production application. For the sake of time, I decided to not spend effort to improve the security of this mechanism.

3.2.3. Publishing a Python Package

Before this project, I had never published any of my python projects to an online repository that enabled other developers to easily incorporate my work into their own projects. In the past, I have used GitHub to store the source code, but sharing source code and sharing an install-able package are quite different. Packages that are easily installed reduce the amount of effort needed by a consumer of the package because they do not have to worry about placing the source code in the python path so that their interpreter can import the package. Furthermore, packages make it easier to receive updates to the code, keep track of different versions, and install dependencies. As for the actual process of organizing my project in order to be distributed as a package, I followed the blog article “How to upload your python package to PyPi” by Joel Barmettler[8]. In his article, Barmettler has his readers create an account for PyPi, which serves as the online repository for the popular python package management tool “pip”. Once a developer has a PyPi account, it is a relatively straightforward process to create releases from the library’s source on GitHub and upload the package to PyPi. Ideally, the entire process of incrementing the version number of the package and preparing the code for deployment would be handled by build scripts. However, for the limited number of releases necessary to publish the package to a working state, I manually performed the steps. The code for the Gold Tree SDK is available on the following GitHub repository: https://github.com/JonScott20/gold_tree_sdk [9].

In addition, a listing of the code be found in Appendix C.
3.3. Data Curation

3.3.1. Exploration

When training a machine learning model, it is important to know which values are predictors for the response variable. The process of selectively using certain data points from available data is known as “feature engineering”. However, knowing which features to include and exclude requires a domain expert. In this section, I examine how different features are correlated with energy production.
It is unsurprising that GHI and energy production are highly correlated. After all, GHI is a measurement of the amount of light energy hitting the surface of the Earth. I find it fascinating the way the two curves mirror each other in the morning and evening, but diverge significantly during the middle of the day. Furthermore, it appears that the maximum energy generated seems to be consistent across the days, regardless of how much extra solar energy there is. I suspect that there must be some limiting factor that prevents the solar system from exactly matching the GHI curve.

![Correlation between GHI and Energy](image1)

**Figure 11: Correlation of GHI and Energy Generation**

![Correlation of Ambient Temperature and Energy Generation](image2)

**Figure 12: Correlation of Ambient Temperature and Energy Generation**
It would appear that the ambient temperature and energy curves also exhibit similar shapes throughout the day. Intuitively this makes sense because the ambient temperature would increase as the sun rises and more of its energy heats up the surface of the planet.

![Figure 13: Correlation of Panel Temperature and Energy Generation](image1)

Unsurprisingly, the temperature of the solar panels also follows a similar curve.

![Figure 14: Comparison of Ambient Temperature and Panel Temperature](image2)

Notice how much warmer the panels are than the surrounding air throughout the course of the day. The panels appear to be roughly 20°C - 30°C
warmer (68°F - 86°F) than the ambient temperature. Many solar panels lose efficiency as they heat up, so it is possible that the increase in panel temperature is acting as a mitigating factor in limiting energy production [10].

I was curious as to what kind of cooling mechanisms could be used to keep the panel temperature in check and I wondered what role the wind could play. In the figure above, the first two days demonstrate how the increasing wind speed seems to align with a decrease in panel temperature. In a simple world, an increase in wind speed would always be helpful in balancing the panel temperature and thus aiding in power generation. However, increases in wind are often associated with changes in weather, changes that may bring clouds and rain to interfere with the power generation.

3.3.2. Interval Selection

There exists production data from the Gold Tree Solar Farm going back to when it first came online in July 2018. However, the sky camera was not operational at that time. In order to use training data that can correspond with images from the sky camera, I have limited my training data to the dates of November 22, 2018 and April 22, 2019. This will give me five months worth of training data. However, it should be noted that this time period is the rainy season for San Luis Obispo, and as such there were many days of rain and thus clouds. Ideally, the training data would be available for multiple years so as to establish a baseline of the seasonal variation. The machine learning models used within this project should improve with a larger quantity of data.

I further narrowed the data set to include data only from 9:00 am to 3:00 pm so that the model will focus on predicting power production during
the time when production should be the highest and so any change in power production would be of the greatest interest. Furthermore, for the winter solstice (December 22), the sun would rise and set in San Luis Obispo at 7:08 am and 4:54 pm. By limiting the window of interest I can avoid images of the sky during the morning and evening in which cloud detection will be different than during the rest of the day. By pulling in the time constraints, I can further focus in on the solar day.

3.3.3. Cleaning

Data collected from a production environment is rarely ever immediately ready for analysis. Missing and nonsensical data can occur from any number of natural causes, from power and network outages to disconnected sensors caused by a nudge from a nearby grazing sheep (this actually happened at the Gold Tree Solar Farm). I found instances in which the wind direction was greater than 360 degrees, the wind speed greater than hurricane force winds, and the power produced for a ten minute period to be greater than the normal production for the entire day. In order to prepare the data that I collected from Gold Tree, I made sure to remove irregularities such as these from the data.

3.4. Sky Image Processing

![Figure 16: Original Sky Camera Image](image)

The sky camera captures details of the surrounding landscape on the outer rim of the circle that encompasses the sky. A close inspection reveals
that some of the solar panels and the nearby hillside are visible. I decided that it would be beneficial to exclude these pixels as a step of the image processing chain. Once an image has been trimmed, it is ready to have its Red-Blue Ratio computed and have the resulting image mapped into two categories with the use of a threshold. The final image consists of an image with two pixel values. White pixels show where clouds have been detected and black pixels denote the absence of clouds.

### 3.4.1. Red Blue Ratio

Calculating the ratio between the red and blue channels of an image is a technique that is frequently used when identifying clouds from an image of the sky [11]. My implementation of calculating the red-blue ratio may be found below.

```python
# Process an image by dividing the red channel by the blue channel.
# Input is expected to by a 2D numpy array.
def process_image_rb_ratio(image):
    rb_ratio = image[..., 0] / image[..., 2]

    # This value was determined by tuning
    threshold = 0.9
    rb_ratio[rb_ratio > threshold] = 255
    rb_ratio[rb_ratio <= threshold] = 0

    # Apply a median filter to smooth edges
    return median(sky_red_blue_ratio, disk(10))
```

Figure 17: Python Code to Compute Red Blue Ratio

The table below demonstrates several examples of the image processing pipeline and the final cloud coverage calculation. It also demonstrates some of the weaknesses of that pipeline. Notice that the sun is falsely identified as a cloud. A full code listing of the image processing pipeline may be found in Appendix D.
3.5. Weatherbit.io

I wanted to be able to feed my models predicted weather data and I decided to use the API provided by Weatherbit.io [12]. I chose this weather API because it provides a 48 hour forecast with the data granularity that I was interested in. Furthermore, it provides estimates for cloud coverage and solar irradiance. The code that I wrote to get forecasts from Weatherbit’s API can be found in Appendix H.

### Table 3: Examples of the Image Processing Pipeline

<table>
<thead>
<tr>
<th>Raw</th>
<th>RBR</th>
<th>Final</th>
<th>Cloud Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Raw Image" /></td>
<td><img src="image2" alt="RBR Image" /></td>
<td><img src="image3" alt="Final Image" /></td>
<td>10.6518</td>
</tr>
<tr>
<td><img src="image4" alt="Raw Image" /></td>
<td><img src="image5" alt="RBR Image" /></td>
<td><img src="image6" alt="Final Image" /></td>
<td>22.0181</td>
</tr>
<tr>
<td><img src="image7" alt="Raw Image" /></td>
<td><img src="image8" alt="RBR Image" /></td>
<td><img src="image9" alt="Final Image" /></td>
<td>48.6936</td>
</tr>
<tr>
<td><img src="image10" alt="Raw Image" /></td>
<td><img src="image11" alt="RBR Image" /></td>
<td><img src="image12" alt="Final Image" /></td>
<td>89.9743</td>
</tr>
<tr>
<td><img src="image13" alt="Raw Image" /></td>
<td><img src="image14" alt="RBR Image" /></td>
<td><img src="image15" alt="Final Image" /></td>
<td>99.14723</td>
</tr>
</tbody>
</table>


3.6. PlantPredict API/SDK

![SDK Documentation Homepage](image)

Figure 18: Screenshot of the PlantPredict SDK Documentation Homepage

PlantPredict exposes API endpoints in order to allow developers to interact with the software. Fortunately, one of the developers from FirstSolar had already put together an alpha version of an SDK for the PlantPredict API [13]. However, the SDK was targeted towards Python 2 and I wanted all of the source code for this project to use a consistent version of Python 3.6. In order to adapt the SDK’s source code into my own project, I was able to download the source code and then update the methods within the source code that were no longer supported in Python 3. This endeavour had the potential to be a large effort, but fortunately, I only needed to modify a single for-loop to use a different type of iteration. A big downside of making this modification is that my project is now dependent upon a branch of the PlantPredict SDK that will not be receiving updates from the developer. I have subscribed to the GitHub repository and have been receiving updates from the developer and from those updates I have learned that a Python 3 version of the SDK is in the works.

3.6.1. Authentication

The SDK requires that a user modify the settings.py file contained within the source code and provide API credentials. API credentials may be generated by PlantPredict account administrator. In my case, Dr. Belanger gave me temporary administrator rights so that I may create the credentials for myself.

3.6.2. Uploading Weather Data

It is important to ensure that all of the weather data that is to be uploaded to PlantPredict has been cleaned of missing data. Incomplete entries in the
data may cause the simulation to fail without indication of the cause. For me, this occurred when I was missing a value for humidity.

3.6.3. Running a Simulation
Once the weather data has been uploaded to the prediction, the prediction may be ran and its results retrieved. In order to access the time-stamped prediction values, it is necessary to drill down into the result object returned by the PlantPredict SDK and locate the “nodal data”. This nodal data can be compared to the actual values recorded by the sensors from the Gold Tree Solar Farm. The code used for uploading weather data and parsing through simulation results for PlantPredict may be found in Appendix F.

3.7. Machine Learning
I built all of my machine learning models using K-Nearest Neighbors regression algorithm. I wrote some rudimentary Python routines to automatically run several iterations of the same feature set in order to select the best k-value for that model. The automation utilized cross validation and focused on minimizing the error on the validation data. All of the code that I used building the models can be found in Appendix G.

4. Evaluation
4.1. Metrics
4.1.1. Root Mean Squared Error (RMSE)
RMSE is a common metric used to evaluate the accuracy of a predictive model on continuous data. It is calculated by first taking the mean of the squared differences between the predicted value and the actual value. Then the square root of the mean is the final step of the calculation in order to convert the units of the measurement back into the units associated with the actual value. The metric gives an indication of the average margin of error for the predictions and is the primary metric used to evaluate models in this project.

```python
# Root Mean Squared Error
def rmse(actual, prediction):
    squared_difference = (
        (actual - prediction) ** 2
    ).dropna()
    return np.sqrt(squared_difference.mean())
```

25
4.1.2. Mean Absolute Percentage Error (MAPE)

MAPE is a common metric used to evaluate the accuracy of a predictive model on continuous data. It represents the average error as a percentage rather than in terms of a certain unit of measure.

```python
# Mean Absolute Percentage Error
def mape(actual, prediction):
    return np.mean(np.abs((actual - prediction) / actual)) * 100
```

4.1.3. Percent Error

Percent error is not a very common metric for evaluating predictive models. I included it because I think it allows me to quickly determine if the model under or over predicts. It is important to note that this is not an average. Rather, it is a percentage calculated by summing the predictions and summing the actual values, and then dividing the sum of predictions by the sum of the actual values.

```python
# Percentage of error between actual
total production and predicted total production.
def percent_error(actual, prediction):
    return (1 - np.abs(prediction.sum() / actual.sum())) * 100
```
4.2. Model Performance

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Expanded</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>Ambient Temperature</td>
</tr>
<tr>
<td>WS</td>
<td>Wind Speed</td>
</tr>
<tr>
<td>WD</td>
<td>Wind Direction</td>
</tr>
<tr>
<td>H</td>
<td>Relative Humidity</td>
</tr>
<tr>
<td>AP</td>
<td>Air Pressure</td>
</tr>
<tr>
<td>GHI</td>
<td>Global Horizontal Irradiance</td>
</tr>
<tr>
<td>DP</td>
<td>Dew Point</td>
</tr>
<tr>
<td>CC</td>
<td>Cloud Coverage</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>PP</td>
<td>PlantPredict</td>
</tr>
<tr>
<td>RMSE</td>
<td>Root Mean Squared Error</td>
</tr>
<tr>
<td>MAPE</td>
<td>Mean Absolute Percentage Error</td>
</tr>
<tr>
<td>PE</td>
<td>Percent Error</td>
</tr>
</tbody>
</table>

Table 4: Acronym Mappings

4.2.1. Overall Model Performance

The table below lists the performance of the models predicting power generation on the training data. The performance recorded in this table should be the best performance we observe because the machine learning models have already seen the power generation values as they were used to train the model. The table denotes several measures of how well the models learned patterns from the training data. An exception should be made for the PlantPredict row, because it was not trained and instead uses complex physics calculations to produce predictions.

<table>
<thead>
<tr>
<th>Features</th>
<th>RMSE</th>
<th>MAPE</th>
<th>PE</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>PP T WS WD H AP GHI DP</td>
<td>100.3654</td>
<td>22.0446</td>
<td>0.0973</td>
<td>6</td>
</tr>
<tr>
<td>ML CC</td>
<td>157.6262</td>
<td>78.9674</td>
<td>-0.1903</td>
<td>8</td>
</tr>
<tr>
<td>ML GHI</td>
<td>43.7970</td>
<td>9.1809</td>
<td>0.0115</td>
<td>3</td>
</tr>
<tr>
<td>ML T WS WD H AP DP</td>
<td>122.1041</td>
<td>57.0588</td>
<td>-0.6930</td>
<td>7</td>
</tr>
<tr>
<td>ML T WS WD H AP GHI DP</td>
<td>46.0665</td>
<td>18.5405</td>
<td>-0.0279</td>
<td>4</td>
</tr>
<tr>
<td>ML T WS WD H AP GHI DP CC</td>
<td>46.7377</td>
<td>19.2922</td>
<td>-0.0935</td>
<td>5</td>
</tr>
<tr>
<td>ML CC GHI</td>
<td>36.2342</td>
<td>7.7603</td>
<td>-0.0028</td>
<td>2</td>
</tr>
<tr>
<td>ML H AP GHI CC</td>
<td>35.1223</td>
<td>11.9205</td>
<td>0.0902</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 5: Performance Predicting on Training Data
The table below shows the performance of the models using data from Gold Tree that was excluded from the training process. In particular, it is the data collected from April 23, 2019 to May 23, 2019. From the viewpoint of the machine learning models, this data is excellent future data. However, the issue with using this data is that we are unable to produce prescient weather forecasts. This table represents how the model could perform if the forecasted data is perfectly accurate. For models that with cloud coverage, this would include images of the sky at a future moment in time.

<table>
<thead>
<tr>
<th>Features</th>
<th>RMSE</th>
<th>MAPE</th>
<th>PE</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>PP T WS WD H AP GHI DP</td>
<td>110.45</td>
<td>11.65</td>
<td>3.02</td>
<td>6</td>
</tr>
<tr>
<td>ML CC</td>
<td>198.72</td>
<td>36.02</td>
<td>30.17</td>
<td>8</td>
</tr>
<tr>
<td>ML GHI</td>
<td>56.27</td>
<td>10.15</td>
<td>-2.75</td>
<td>4</td>
</tr>
<tr>
<td>ML T WS WD H AP DP</td>
<td>184.13</td>
<td>48.37</td>
<td>2.36</td>
<td>7</td>
</tr>
<tr>
<td>ML T WS WD H AP GHI DP</td>
<td>67.76</td>
<td>13.70</td>
<td>-1.95</td>
<td>5</td>
</tr>
<tr>
<td>ML T WS WD H AP GHI DP CC</td>
<td>56.01</td>
<td>11.63</td>
<td>0.19</td>
<td>3</td>
</tr>
<tr>
<td>ML CC GHI</td>
<td>45.39</td>
<td>8.01</td>
<td>-1.18</td>
<td>1</td>
</tr>
<tr>
<td>ML H AP GHI CC</td>
<td>47.86</td>
<td>8.60</td>
<td>0.97</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 6: Performance Predicting on Data Excluded from Training

The results from Table 5 and Table 6 suggest that the machine learning model that uses cloud coverage and GHI is the best model. This model does a good job of predicting power generation working under the assumption that we can provide extremely accurate forecasts for cloud coverage and GHI. The next table shows an average of model performance on the entire 48 hour forecast provided by Weatherbit.

<table>
<thead>
<tr>
<th>Features</th>
<th>RMSE</th>
<th>MAPE</th>
<th>PE</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>PP T WS WD H AP GHI DP</td>
<td>212.58</td>
<td>53.52</td>
<td>32.52</td>
<td>8</td>
</tr>
<tr>
<td>ML CC</td>
<td>186.67</td>
<td>40.73</td>
<td>23.50</td>
<td>4</td>
</tr>
<tr>
<td>ML GHI</td>
<td>205.58</td>
<td>54.27</td>
<td>32.43</td>
<td>7</td>
</tr>
<tr>
<td>ML T WS WD H AP DP</td>
<td>158.59</td>
<td>46.94</td>
<td>19.40</td>
<td>1</td>
</tr>
<tr>
<td>ML T WS WD H AP GHI DP</td>
<td>169.84</td>
<td>46.60</td>
<td>21.78</td>
<td>3</td>
</tr>
<tr>
<td>ML T WS WD H AP GHI DP CC</td>
<td>168.81</td>
<td>44.11</td>
<td>20.68</td>
<td>2</td>
</tr>
<tr>
<td>ML CC GHI</td>
<td>197.96</td>
<td>51.01</td>
<td>30.11</td>
<td>5</td>
</tr>
<tr>
<td>ML H AP GHI CC</td>
<td>198.72</td>
<td>50.90</td>
<td>29.67</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 7: Performance Predicting on Forecasted Data provided by Weatherbit.io
4.2.2. Short Term Model Performance

The following graphs display the predictions generated using the machine learning model with GHI and cloud coverage as features using the Weatherbit forecast and the weather data later collected at Gold Tree.

Table 8: Validation of Weatherbit Forecasts

In the left column, observe how poorly the curves match up to one another. The actual power production demonstrates that these days had periods where production varied greatly within short amounts of time. The prediction failed to capture these variations. In the right column, observe
that the prediction better follows the overall shape of the production curve in comparison to the charts that use the Weatherbit data. I believe this to be evidence that using Weatherbit data as future data is not a good way to get accurate predictions.

5. Conclusions and Future Work

5.1. Conclusion

5.1.1. Incomplete Goal of Short Term Prediction

The majority of my effort for this project went into developing the data infrastructure and image processing. As a consequence, I was unable to generate viable future data to feed into the models. I attempted to feed my models data returned by a weather forecasting API, but found disappointing results for short-term and overall prediction accuracy. I believe the primary reason for the failure of this technique to be attributed to the fact that the cloud coverage and GHI estimates were not for the same coordinates as the Gold Tree facility. This is important because of the direct influence of GHI on power generation. In the future work section, I discuss my ideas for what could be done to obtain future data conducive to short term predictions.

5.1.2. PlantPredict and Operational Forecasting

I do not think that PlantPredict should be used for short term power prediction. The tool is developed to aid in the planning process of building a solar facility and was not designed for short term power prediction. Most of my machine learning models out performed PlantPredict on weather data collected by the Gold Tree facility. I believe that machine learning and similar methods will be better at short term power predictions because they can learn patterns from the specific facilities.

5.1.3. Alternate Application

I was able to train a machine learning model that works at predicting energy generation on data that was not included in the training process. This performance is attributable to the fact that the sources of data are the same in for the training and prediction process. For this reason, I suspect that my machine learning model could actually be used for something other than prediction specifically system health. I believe that my model could be decomposed into smaller models than represent the inverters instead of the site as a whole. In this scenario, each model would serve as a watchdog on
the inverters’ performance and could raise an alert whenever performance is outside of the expected performance given environmental conditions.

5.2. Future Work

5.2.1. Improved Image Processing

The images captured by the sky camera are distorted because of the fish eye lens of the camera. This means that the size of the clouds on the perimeter of the image are misrepresented. There are techniques for correcting this distortion, but the one I wanted to attempt required physical access to the camera [11][14].

The technique I used to determine cloud coverage could be refined further. As we see from table 9, the very first image is a pretty clear day, but the cloud coverage is still calculated to be 10.65% because the sun is falsely identified as a cloud. This could be corrected by generating clear sky images and using them with the captured images in order to filter out the interference from the sun [11].

<table>
<thead>
<tr>
<th>Raw</th>
<th>RBR</th>
<th>Final</th>
<th>Cloud Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Raw Image" /></td>
<td><img src="image2.png" alt="RBR Image" /></td>
<td><img src="image3.png" alt="Final Image" /></td>
<td>10.6518</td>
</tr>
</tbody>
</table>

Table 9: Over Approximation of Cloud Coverage

I would have liked to try to predict a future sky camera image given the recent images. This would have allowed me to predict the future locations of clouds as they move across the sky [11]. The techniques for calculating cloud shadow already have ongoing research, but I was unable to pursue this because of the large gap between images from the sky camera. When the clouds are moving quickly across the sky, I wasn’t able to manually trace cloud movements between successive images. Observe the table below as a demonstration of the difficulty in tracing cloud movement.
I do not know the minimum frequency at which the camera will need to take pictures in order for clouds to be traceable. However, I do know that an increase in capture frequency will increase the cost of the camera hosting repository. The camera currently captures images all day and night. Unless there exists a research initiative in energy production from lunar energy, I recommend that the camera be configured to not capture images of the night sky. This way, the storage space that would have been used to store the images of the night sky could be used for a double increase in daytime capture frequency.

5.2.2. Alternative Data Source

This project has focused on looking at the sky from the ground. However, in the modern era, there are a plethora of satellites capturing images of the Earth from space. I think it would be interesting to tap into this information. I found that the National Weather Service does make satellite imagery available to the public [15] [16].

5.2.3. Alternative Models

At the start of this project, I was only familiar with the machine learning techniques that I had learned in a single quarter of my Introduction to Data Science class. Since then, I’ve taken a class on Deep Learning and I believe that the use of the techniques within Deep Learning could be helpful in increasing the accuracy of predictions. In particular, I would recommend that convolutional and recurrent neural networks be investigated. Convolutional networks tend to work well on processing image data and could be targeted towards the sky camera imagery in two capacities. One use could be for cloud segmentation, similar to the way I calculate cloud coverage. The other use could be for generating a future image of the sky. Recurrent networks work well on problems where the data is sequential in nature such as time series forecasting [4][7].
5.3. Continuous Training and Weighted Recent Data

The machine learning models were trained on strict set of data, November 22, 2018 through April 22, 2019. A more reliable, industry scale prediction solution would not train on a fixed set of data. It would need to be continuously retrained on all of the data collected up to the present moment. I suspect there may be a benefit to weighting the most recent data more heavily than the historical data. This weighting could be a good way to counteract the soiling of the solar panels or sensors.

References


6. Appendices

Appendix A. Summary of Learning

I am grateful for everything that I have learned in my experience working on this project. The project provided me with the opportunity to learn about solar power, a topic that I knew very little about before the project. My abilities as a Cal Poly computer science student were put to the test. The hardest part of the project was not like the difficult parts of my other computer science classes. The challenge of my other classes was from the technical challenges of implementing the algorithms in the assignments. For this project, the biggest challenges were managing myself, staying organized, and working diligently. It was challenging to integrate the separate components of the project together. I believe I tried to refactor the code three or four times throughout the project in order to make it more manageable. I’m still not entirely satisfied with the organization of it all. I suppose that is a lesson in itself about the reality of a being professional developer. I find it even more rewarding that I was able to publish a part of my work on the Gold Tree SDK so that other students could use it.

Appendix B. Code Listing Note

The code listings do not include many of the import statements required for them to run. They are listed here as demonstration of the work that was completed. If you are interested in running any of the code for your own use, please reach out to me at jonscott20@gmail.com.

Appendix C. Gold Tree SDK

class RequestManager:
    API_ENDPOINT = (  
        'https://webapidukerec.horizon' +  
        '.greenpowermonitor.com/'
    )
    FACILITY_ID = 42
    PRODUCTION_IDS = {
        "29373": "Energy (kWh)"
    }
    SPECIFIC_POWER_ID = "29380"

35
WEATHER_IDS = weather_ids = {
    '27986': 'Plant Irradiance (GHI) W/m^2',
    '27985': 'Plant Irradiance (POA) W/m^2',
    '27987': 'Panel Temperature (C)',
    '27984': 'Ambient Temperature (C)',
    '27993': 'Wind Speed (m/s)',
    '27990': 'Relative Humidity (%)',
    '27983': 'Air Pressure (hPa)',
    '24544': 'Dew Point (C)',
    '27992': 'Wind Direction ( )'
}

GROUPING_MODE = [
    "raw",
    "quarter",
    "day",
    "month",
    "year",
    "hour",
    "halfyear",
    "tenminute"
]

def __init__(self):
    self.authentication_status = "Unauthenticated"
    self.accessToken = self.authenticate()

def authenticate(self):
    url = self.API_ENDPOINT + 'api/Account/Token'
    username = ''
    password = ''
    body = {'username': username, 'password': password}
    data = parse.urlencode(body).encode()
    req = request.Request(url, data=data)
    resp = request.urlopen(req)
    result = json.load(resp)
    accessToken = result["AccessToken"]
    return accessToken
def get_kpi(self, target_kpi, group_mode, start_date, end_date):
    url = ('{api_endpoint}api/DataList' + '?datasourceId={datasource_id}' + '&startDate={start_date}' + '&endDate={end_date}' + '&aggregationType={agg_type}' + '&grouping={grouping}').format(api_endpoint=self.API_ENDPOINT, datasource_id=target_kpi, start_date=start_date, # 1547942400, end_date=end_date, # 1548028800, agg_type=0, grouping=group_mode)
    req = request.Request(url)
    req.add_header('Authorization', 'Bearer ' + self.accessToken)
    resp = request.urlopen(req)
    return resp

def get_power_production_data(self, start_date, end_date, grouping_mode):
    new_df = pd.DataFrame()
    for id, name in self.PRODUCTION_IDS.items():
        temp_df = pd.read_json(
            self.get_kpi(id, grouping_mode, start_date, end_date)
        )
        temp_df[name] = temp_df["Value"]
        temp_df = temp_df.set_index('Date').drop(
            columns=['DataSourceId', 'Value'])
if new_df.equals(pd.DataFrame()):
    new_df = temp_df.copy(deep=True)
else:
    new_df[name] = temp_df[name]
    # A sleep to go be kind to the GPM API
    time.sleep(0.5)
return new_df

def get_historical_weather_data(self, start_date, end_date, grouping_mode):
    new_df = pd.DataFrame()
    for id, name in self.WEATHER_IDS.items():
        temp_df = pd.read_json(
            self.get_kpi(id, grouping_mode, start_date, end_date)
        )
        temp_df[name] = temp_df["Value"]
        temp_df = temp_df.set_index('Date').drop(
            columns=['DataSourceId', 'Value'])
    if new_df.equals(pd.DataFrame()):
        new_df = temp_df.copy(deep=True)
    else:
        new_df[name] = temp_df[name]
    # A sleep to go be kind to the GPM API
    time.sleep(0.5)
return new_df

@staticmethod
def eprint(*args, **kwargs):
    print(*args, file=sys.stderr, **kwargs)

@staticmethod
def clean_data(training_data):
    # Drop all the observations with at
    # least one piece of invalid data.
    training_data = training_data.dropna()

    # Remove all the observations where
    # energy generation is 0.
    training_data = (training_data.loc[
        training_data["Energy (kWh)"] != 0
    ])

    # Remove all the observations where
    # the energy generation is greater
    # than 1000. These observations occur
    # when a two or more observations are combined.
    training_data = (training_data.loc[
        training_data["Energy (kWh)"] < 1000
    ])

    # Drop all observations with a wind
    # direction greater than 360
    training_data = training_data.loc[
        training_data["Wind Direction ( )"] < 360]

    # Drop all observations with a wind
    # speed of greater than 70 m/s
    training_data = training_data.loc[
        training_data["Wind Speed (m/s)"] < 70]

    return training_data

def generate_training_data(self, start, end, groupingMode):
    """
    :param start: yyyy-mm-dd
    :param end: yyyy-mm-dd
    :param groupingMode: normally "tenminute"
    """
return: a dataframe of weather and power data

big_dividing_factor = 1000000000

# Convert the start and end strings
# to datetime indexes
dates = pd.to_datetime([start, end])
start_date = int(dates[0].value / big_dividing_factor)
end_date = int(dates[1].value / big_dividing_factor)

# Query the API for data
power_df = self.get_power_production_data(
    start_date, end_date, groupingMode
)
weather_df = self.get_historical_weather_data(
    start_date, end_date, groupingMode
)
training_df = pd.concat([power_df, weather_df], axis=1)
return self.clean_data(training_df)

@staticmethod
def generate_file_name(start_date, end_date):
    start_date_time = datetime.datetime.strptime(start_date, '%m/%d/%Y')
    end_date_time = datetime.datetime.strptime(end_date, '%m/%d/%Y')
    start_string = "{:m-%d-%Y}".format(start_date_time)
    end_string = "{:m-%d-%Y}".format(end_date_time)
def generate_power_production_file_name(self, start_date, end_date, extension):
    return ('GOLD_TREEPOWER_
           + self.generate_file_name(
               start_date,
               end_date
           )
           + extension)

def generate_weather_file_name(self, start_date, end_date, extension):
    return ('GOLD_TREEWEATHER_
           + self.generate_file_name(
               start_date,
               end_date
           )
           + extension)

Appendix D. Image Processing Pipeline

# Image Processing
# Trim out the extra black on the left and
# right side of the image
def trim_outer_circle(sky):
    x1_trim = 300
    x2_trim = 2280

    # Determine the dimensions of the image
    width = sky.shape[0]
    height = sky.shape[1]
    center_on_width = int(width / 2)
    center_on_height = int(height / 2)
    circle_radius = 971

    # Trim the pixels around the sky
sky_circle = np.zeros((width, height), dtype=np.uint8)
rr, cc = draw.circle(
    center_on_width, center_on_height, circle_radius
)
sky_circle[rr, cc] = 1
sky_trimmed = sky.copy()
sky_trimmed[sky_circle == 0] = 0
return sky_trimmed[:, x1_trim: x2_trim]

# Convenience function that loads an image
# and trims the outer circle
def load_and_trim(filename):
    sky = io.imread(filename)
    sky_trimmed = trim_outer_circle(sky)
    return sky_trimmed

# Process an image by dividing the red channel
# by the blue channel
def process_image_rb_ratio(sky_trimmed):
    ratio = (sky_trimmed[..., 0] / sky_trimmed[..., 2])

    # This value was determined by tuning
    threshold = 0.9
    ratio[ratio > threshold] = 255
    ratio[ratio <= threshold] = 0

    # Apply a median filter to smooth edges
    return median(ratio, disk(10))

# Determine the percentage of the sky covered
# by clouds by counting pixels
def calculate_cloud_cover(image, pixels_in_circle):
    white = 255

    # image is currently a 2D array, convert to 1D
flattened_image = image.flatten()
white_pixels = np.sum(flattened_image == white)

# White pixels denote areas where clouds have been detected
cloud_percentage = (
    white_pixels / pixels_in_circle * 100
)
return cloud_percentage

# Convenience method to completely process an image from its filepath
def cloud_cover_from_image(image_path, pixels_in_circle):
    image = load_and_trim(image_path)
    rbr_image = process_image_rb_ratio(image)
    coverage = calculate_cloud_cover(
        rbr_image, pixels_in_circle
    )
    return coverage

# Process an image and save it
def process_and_save_image(target_date, filename):
    image = load_and_trim(filename)
    outfile = target_date + '-' + filename

    rbr_output_filename = (
        settings.rbr_output_directory + outfile
    )
    rbr_image = process_image_rb_ratio(image)
    io.imsave(rbr_output_filename, rbr_image)

    rbdiff_output_filename = (
        settings.rbdiff_output_directory + outfile
    )
    rbdiff_image = process_image_rb_difference(image)
    io.imsave(rbdiff_output_filename, rbdiff_image)
def process_directory(target_date, target_data_directory):
    sky_image_filenames = [
        f for f in listdir(target_data_directory)
        if isfile(join(target_data_directory, f))
    ]
    for sky_image_filename in sky_image_filenames:
        process_and_save_image(
            target_date,
            target_data_directory + sky_image_filename)
        print("Completed: " + sky_image_filename)

def create_coverage_tuple(image_path, pixels_in_circle):
    # strip file extension
    time_str = image_path.parts[-1].split('.')[0]
    try:
        coverage = cloud_cover_from_image(
            image_path, pixels_in_circle
        )

        # Create a tuple of local timestamp
        # and cloud coverage
        result = (time_str, coverage)
    return result
    except ValueError:
        return None

def estimate_completion_time(directory_path, completed):
    process_time = 1
    image_count = len(listdir(directory_path)) - completed
    hour_conversion = 3600
    estimated_time = (
        process_time * image_count / hour_conversion
    )
    print("Remaining time {0} hours. Completed {1}".format(
        estimated_time, completed))

# Process all the images in a directory

def cloud_coverage_from_directory(directory_path):
    circle_radius = 971
    pixels_in_circle = int(np.pi * (circle_radius ** 2))
    coverages = []
    images = listdir(directory_path)
    images.sort()
    for i, file in enumerate(images):
        file_path = Path(str(directory_path) + "/" + file)
        if i % 50 == 0:
            estimate_completion_time(directory_path, i)
            save_coverages(coverages, i)
            coverages = []
        coverage = create_coverage_tuple(
            file_path, pixels_in_circle
        )
        coverages.append(coverage)
    print("Finished")

def save_coverages(coverages, i):
    pst = pytz.timezone("America/Los_Angeles")
    cloud_df = pd.DataFrame(
        coverages, columns=["Date", "Cloud Coverage"]
    )
    cloud_df.Date = pd.to_datetime(cloud_df.Date, utc=True)
    cloud_df.set_index("Date", inplace=True)
    cloud_df["Cloud Coverage"] = pd.to_numeric(
        cloud_df["Cloud Coverage"]
    )

    # Sort by the time index
    cloud_df = cloud_df.sort_index()

    # Interpolate to a minute frequency
    cloud_df = cloud_df.reindex(
        cloud_df.index.tz_convert(pst)
    )

    # Export dataframe
Appendix E. Downloading Sky Images

# Image Retrieval
# Download sky camera images from the web hosted repository

def download_sky_camera_images(start_time_string, end_time_string):
    # Winter solistace (December 22) as sunrise
    # and sunset as 7:08am and 4:54pm.
    # I have chosen to move the window of interest
    # in a couple hours in order to focus on the
    # "solar day"

    # 9:00 am
    start_window = '09:00:00 -0700'

    # 3:00 pm
    end_window = '15:00:00 -0700'

    window_parse = '%H:%M:%S %z'
    start_window_datetime = datetime.datetime.strptime(
        start_window, window_parse
    ).time()
    end_window_datetime = datetime.datetime.strptime(
        end_window, window_parse
    ).time()

    # Concatenate the date and time together
    start_datetime_str = "{0} {1}".format(
        start_time_string, start_window
    )
    end_datetime_str = "{0} {1}".format(
        end_time_string, end_window
    )

    # Construct datetime objects
parse_format = '%Y-%m-%d %H:%M:%S %z'
start_datetime_obj = datetime.datetime.strptime(
    start_datetime_str, parse_format
)
end_datetime_obj = datetime.datetime.strptime(
    end_datetime_str, parse_format
)

# How frequently the images are available for download.
# This could increase if REC lets it last info
# says they asked for a quote to increase it.
capture_interval = 5

# Where the images are available for download
image_url = (
    'https://cameraftpapi.drivehq.com/api/
    + 'Camera/GetLastCameraImage.aspx?
    + 'parentID=229229999&shareID=14125452'
    + '&time='
) polite_delay = 0.5

current_datetime_obj = start_datetime_obj
while current_datetime_obj < end_datetime_obj:
curr_time = current_datetime_obj.time()
if (curr_time < start_window_datetime
    or curr_time > end_window_datetime):
    print("Ignoring", curr_time)
else:
    # download the image
    filename = str(current_datetime_obj) + '.jpg'
    utc_datetime = current_datetime_obj.astimezone(
        pytz.utc
    )
    print(utc_datetime)
    output_path = Path(str(settings.data_directory)
    + '/skycam/' + filename)
    request_url = image_url.format(
        time=utc_datetime.strftime(
            '%Y-%m-%d%H:%M:%S'
        )
    )
def retry_retrieve(request_url, output_path):
    retry_count = 0
    retry_max = 5

    print(output_path)
    while retry_count < retry_max:
        try:
            resp = urllib.request.urlopen(request_url)
            with open(output_path, 'wb') as f:
                f.write(resp.read())
            break
        except urllib.error.URLError as e:
            print("EXCEPTION: " + request_url + " " + str(e))
            continue
        except urllib.error.HTTPError as e:
            print("EXCEPTION: " + request_url + " " + str(e))
            continue
        except socket.error as e:
            print("EXCEPTION: " + request_url + " " + str(e))
            continue
        time.sleep(retryDelay)
        retry_count += 1
        if retry_count == retry_max:
            print("Unable to retrieve image. Moving on...")

# Reattempt to download image if error occurs

# Expects date format of YYYY-MM-DD as command line args

def main(argv):
    48
start = argv[0]
end = argv[1]
download_sky_camera_images(start, end)

# Standard boilerplate to call the main() function to begin
# the program.
if __name__ == '__main__':
    main(sys.argv[1:])

Appendix F. Use of PlantPredict SDK

# PlantPredict Prediction Methods
def pp_upload_weather_forecast(weather_df):
    location_info = plantpredict.Geo.get_location_info(
        latitude=settings.gtsf_latitude,
        longitude=settings.gtsf_longitude
    )
    # Map the columns to the name that the
    # PlantPredict API expects
    weather_df["time_stamp"] = weather_df.index.strftime(
        '%Y-%m-%dT%H:%M:%S')
    weather_df.rename(columns={
        "Ambient Temperature (°C)" : "temperature",
        "Plant Irradiance (GHI) W/m²" : "global_horizontal_irradiance",
        "Relative Humidity (%)" : "relative_humidity",
        "Wind Speed (m/s)" : "windspeed",
        "Air Pressure (hPa)" : "pressure",
        "Dew Point (°C)" : "dew_point"
    }, inplace=True)

    # Add the index column as expected by
    # the PlantPredict API
    weather_df["index"] = range(0, len(weather_df))

    # Drop unused columns
    weather_df.drop(columns=[
        # "Plant Irradiance (POA) W/m²",
        # "pressure",
    ],
# "windspeed",
# "relative_humidity",
# "Dew Point (C)",
# "Precipitation (mm)",
"Wind Direction ()",
# "Energy (kWh)"
], inplace=True)

weather_list = []
weather_df.apply(lambda x:
    (weather_list.append(x.to_dict())), axis=1)

weather = plantpredict.Weather()
weather.name = "Gold Tree Automated Weather Data"
weather.latitude = settings.gtsf_latitude
weather.longitude = settings.gtsf_longitude
weather.country = location_info['country']
weather.country_code = (
    location_info['country_code']
)
weather.data_provider = (
    weather_data_provider_enum.OTHER
)

# this is the weather data file
weather.weather_details = weather_list
weather.elevation = round(
    plantpredict.Geo.get_elevation(
        latitude=settings.gtsf_latitude,
        longitude=settings.gtsf_longitude
    )['elevation'], 2)
weather.locality = location_info['locality']
weather.region = location_info['region']
weather.state_province = (
    location_info['state_province']
)
weather.state_province_code = (
    location_info['state_province_code']
)
weather.time_zone = plantpredict.Geo.get_time_zone(50
latitude=settings.gtsf_latitude,
longitude=settings.gtsf_longitude
)['timezone']
weather.status = library_status_enum.DRAFT_PRIVATE
weather.data_type = weather_data_type_enum.MEASURED
weather.p_level = weather_plevel_enum.P95
weather.time_interval = 10  # minutes
weather.global_horizontal_irradiance_sum = round(
    sum([w['global_horizontal_irradiance']
        for w in weather_list]) / 1000, 2
)
weather.average_air_temperature = np.round(
    weather_df['temperature'].mean(), 2)
weather.average_relative_humidity = np.round(
    weather_df['relative_humidity'].mean(), 2)
weather.average_windspeed = np.round(
    weather_df['windspeed'].mean(), 2)
weather.max_air_temperature = np.round(
    weather_df['temperature'].max(), 2)
weather.create()

return weather

# Update the target prediction to use the
# newly updated weather file
def pp_update_prediction(prediction, weather):
    prediction.start_date = weather.start_date
    prediction.end_date = weather.end_date
    prediction.start = weather.start_date
    prediction.end = weather.end_date
    prediction.weather_id = weather.id
    prediction.update()
    return prediction

# Run a PlantPredict prediction
def pp_run_prediction(prediction):
    export_options = {
        'export_system': True,
'block_export_options': [{
    "name": 1,
    "export_block": False,
    "export_arrays": False,
    "export_inverters": False,
    "export_dc_fields": False
}]
}
prediction.run(export_options=export_options)
return prediction

def pp_save_prediction(system_data):
    system_df = pd.DataFrame(system_data)

    # Value is in terms of W. So convert
    # it to kW by dividing by 1000
    # then to kWh by multiplying by 1/6
    # of an hour (aka ten minutes)
    system_df["energy_prediction"] = (
        system_df["plant_power_generated"] * (1 / 6) / 1000
    )

    system_df["Date"] = pd.to_datetime(
        system_df["timestamp"]
    )
    system_df.set_index("Date", inplace=True)

    start_time = str(system_df.index[0])
    end_time = str(system_df.index[-1])
    time_file_name = sputil.GenerateFileNameFromDate(
        start_time,
        end_time,
        "%Y-%m-%d %H:%M:%S"
    )

    output_file_name = (time_file_name + ".csv"
)
    print("" + output_file_name)
output = system_df["energy prediction"]
output.to_csv(
    output_file_name, header=["Energy (kWh)"]
)
return output

def pp_generate_prediction(weather_df):
    # Authenticate
    result = plantpredict.OAuth2.token()
    if result.status_code == 200:
        print("Successful authentication" + " with PlantPredict")
    else:
        print("Failed authentication with PlantPredict")
    return

    # Get the prediction of interest
    project_id = 12489
    prediction_id = 128204
    prediction = plantpredict.Prediction(
        id=prediction_id,
        project_id=project_id
    )

    print("Uploading weather forecast...")
    weather = pp_upload_weather_forecast(
        weather_df.copy(deep=True)
    )
    print("Weather uploaded."

    # Update the prediction to use the new weather file
    weather.get()
    prediction.get()
    pp_update_prediction(prediction, weather)
    print("Prediction updated."

    # Run the prediction
print("Running prediction...")
pp_run_prediction(prediction)
print("Prediction complete")

# Save the results
system = prediction.get_nodal_data(
    params={'system': True}
)
print("Saving prediction...")
prediction = pp.save_prediction(system['system_data'])
print("Prediction saved.")
return prediction

Appendix G. Machine Learning

# Machine Learning Prediction Methods
ModelResult = collections.namedtuple(
    'ModelResult',
    'File Label Prediction Best_K Best_RMSE Model'
)

def ml_rmse_given_feature_set(features, k, x_train, y_train):
    x_dict = x_train[features].to_dict(orient="records")
    # specify the pipeline
    vec = DictVectorizer(sparse=False)
    scalar = preprocessing.StandardScaler()
    model = KNeighborsRegressor(n_neighbors=k)
    pipeline = Pipeline([("vectorizer", vec),
                         ("scaler", scalar),
                         ("fit", model)])
    
    scores = cross_val_score(
        pipeline, x_dict, y_train, cv=10,
        scoring="neg_mean_squared_error",
        error_score=np.nan
    )
return np.sqrt(np.mean(-scores))

def ml_build_model_given_best(training_data_path, feature_set, best_k):
    # Check to see if we have run created a model
    # for this training data before
    # If we have, use the parameters that were
    # determined the best.
    training_data = pd.read_csv(training_data_path)
    training_data.set_index("Date", inplace=True)
    training_data_features = training_data[feature_set]
    training_data_label = training_data["Energy (kWh)"
    x_dict = training_data_features.to_dict(
        orient="records"
    )
    vec = DictVectorizer(sparse=False)
    scalar = preprocessing.StandardScaler()
    model = KNeighborsRegressor(n_neighbors=best_k)
    pipeline = Pipeline([(
        "vectorizer", vec),
        ("scaler", scalar),
        ("fit", model)])
    pipeline.fit(x_dict, training_data_label)
    return pipeline

def ml_build_model(training_data, feature_set):
    training_data_features = training_data[feature_set]
    training_data_label = training_data["Energy (kWh)"
    k_min = 1
    k_max = 25
    k_step = 1
    ks = pd.Series(range(k_min, k_max, k_step))
    ks.index = range(k_min, k_max, k_step)
    validation_errs = ks.apply(
        lambda x: ml_rmse_given_feature_set (feature_set,
                                             x,
                                             training_data,
                                             training_data_label
    )
best_model = validation_errs.sort_values(
    ascending=True
).head(1)
best_k = list(best_model.index.values)[0]
best_rmse = list(best_model.values)[0]

x_dict = training_data_features.to_dict(
    orient="records"
)

# specify the pipeline
vec = DictVectorizer(sparse=False)
scalar = preprocessing.StandardScaler()
model = KNeighborsRegressor(n_neighbors=best_k)
pipeline = Pipeline(["vectorizer", vec],
                    ["scaler", scalar],
                    ["fit", model])
pipeline.fit(x_dict, training_data_label)
predictions = pipeline.predict(x_dict)
prediction_series = (pd.Series(predictions,
                            index=training_data_label.index.values))

return ModelResult(
    File = None,#training_data_path ,
    Label = training_data_label ,
    Prediction = prediction_series ,
    Best_K = best_k,
    Best_RMSE = best_rmse ,
    Model = pipeline
)

def ml_model_summary(model_result):
    model_result.Label.plot.line(legend=True)
    (model_result.Prediction.plot.line(legend=True)
    .legend(["Produced", "Predicted"]))
produced_sum =  model_result.Label.sum().round(2)
predicted_sum = model_result.Prediction.sum().round(2)
start_month = (}
```python
datetime.datetime.strptime(
    model_result.Label.index[0],
    "%Y-%m-%d %H:%M:%S") . strftime("%B")
end_month = (datetime.datetime.strptime(
    model_result.Label.index[-1],
    "%Y-%m-%d %H:%M:%S") . strftime("%B"))

print("Starting Month: "+start_month)
print("Ending Month: "+end_month)
print("Best K: "+str(model_result.Best_K))
print("Best RMSE: "+str(model_result.Best_RMSE))
print("Produced Energy (MWh): "+str(produced_sum / 1000))
print("Machine Learning Predicted Energy (MWh): "+str(predicted_sum / 1000))
print("Difference (MWh): "+str((predicted_sum - produced_sum) / 1000))
print("Percent Difference (%): "+str(((predicted_sum / produced_sum) - 1) * 100))

def ml_run_prediction(model, weather_forecast):
    print("Running machine learning prediction...")
    future_data = model.predict(
        weather_forecast.to_dict(orient="records")
    )
    future_data_series = (pd.Series(
        future_data,
        index=weather_forecast.index.values
    ))
    future_data_series.index.name = "Date"
    print("Finished running machine"
        + "learning prediction..."
    )
```

57
return future_data_series

def ml_save_prediction(prediction_series, model_type):
    print("Saving machine learning prediction...")
    start_time = str(prediction_series.index[0])
    end_time = str(prediction_series.index[-1])
    time_file_name = sputil.GenerateFileNameFromDate(
        start_time,
        end_time,
        '%Y-%m-%d %H:%M:%S'
    )
    output_file_name = (  
        settings.machine_learning_prediction_directory  
        + time_file_name + '.csv'
    )
    print(" " + output_file_name)
    prediction_series.to_csv(  
        output_file_name,  
        header=['Energy (kWh)']
    )
    print("Machine learning prediction saved.")

Appendix H. Weatherbit Forecast

# Get the current Weatherbit forecast for the Gold Tree Solar Farm
def get_weatherbit_forecast():
    attribute_name_map = {
        'timestamp_local': 'Date',
        'dhi': 'DHI (Clear Sky) (W/m^2)',
        'dni': 'DNI (Clear Sky) (W/m^2)',
        'ghi': 'GHI (Clear Sky) (W/m^2)',
        'solar_rad': 'Estimated Solar Radiation (W/m^2)',
        'wind_dir': 'Wind Direction',
        'wind_spd': 'Wind Speed (m/s)',
        'temp': 'Ambient Temperature (C)',
        'dewpt': 'Dew Point (C)',
        'rh': 'Relative Humidity (%)',
        'pres': 'Air Pressure (hPa)',
        'clouds': 'Cloud Cover (%)',
    # Historical data doesn’t contain these metrics
# 'clouds_low': 'Low Cloud Cover (%)',
# 'clouds_mid': 'Mid Cloud Cover (%)',
# 'clouds_hi': 'High Cloud Cover (%)',
'vis': 'Visibility (kilometers)'
}

coordinates_request_url = (
    'https://api.weatherbit.io/v2.0/forecast/hourly' +
    '?lat=%s&lon=%s&key=%s' % (settings.gtsf_latitude,
                               settings.gtsf_longitude,
                               settings.weatherbit_api_key)
)
resp = requests.get(coordinates_request_url).json()
forecast_df = json_normalize(resp['data'])
forecast_simple = (  
    forecast_df[
        list(attribute_name_map.keys())
    ].copy(deep=True)
)
forecast_simple.rename(
    columns=attribute_name_map, inplace=True
)

start_time = forecast_simple.Date.values[0]
end_time = forecast_simple.Date.values[-1]
file_time_name = sputil.GenerateFileNameFromDate(  
    start_time,
    end_time,
    '%Y-%m-%dT%H:%M:%S'
)

forecast_simple['Date'] = pd.to_datetime(forecast_simple['Date'])
forecast_simple.set_index('Date', inplace=True)
weatherbit_output_file_name = (  
    str(settings.weather_forecast_data_directory) +
    '/WEATHERBIT_FORECAST' +
    str(file_time_name) +
    '.csv'
)
print(weatherbit_output_file_name)
# Save the forecasted data to file

forecast_simple.to_csv(weatherbit_output_file_name)

# Return the forecast as a dataframe
return forecast_simple

# Get the latest weather forecast and interpolate it.
def get_latest_forecast():
    weather_forecast = get_weatherbit_forecast()
    model_columns_map = {
        "Date": "Date",
        ("Estimated Solar Radiation (W/m^2)"
            : "Plant Irradiance (GHI) W/m^2"),
        "Temperature (C)": "Ambient Temperature (C)",
        "Wind Speed (m/s)": "Wind Speed (m/s)",
        "Wind Direction": "Wind Direction ()",
        "Relative Humidity (%)": "Relative Humidity (%)"
    }
    weather_forecast.rename(
        columns=model_columns_map, inplace=True
    )
    weather_forecast.drop(columns=[
        'DHI (Clear Sky) (W/m^2)',
        'DNI (Clear Sky) (W/m^2)',
        'GHI (Clear Sky) (W/m^2)',
        'Dew Point (C)',
        'Air Pressure (hPa)',
        'Cloud Cover (%)',
        'Visibility (kilometers)' ]
    , inplace=True)

    # Interpolate hourly forecast to a
ten minute granularity
    weather_forecast = (  
        weather_forecast.resample('10T').asfreq().interpolate()
    )

    return weather_forecast

60