373 FINAL PROJECT REPORT

(Fall 2018)

Daniel Hu (hu432), Yash Pujara (ypujara),
Nirali Rai (rai15), and Jessica Tian(tian99)

Final Report

Problem:

Given various parameters such as temperature or light level, we want to be able to predict whether an office room is occupied. If we can accurately predict occupancy in a room, we can therefore cut down on energy costs (i.e. why turn the heat/A.C. up when there is no one in the room or keeping the lights on)

Dataset:

This dataset contains different samples, seven different attributes: date/time (Current date in year-month-day hour:minute:second format), temperature (Celsius), relative humidity (percentage), light (lux), CO2 (ppm), humidity ratio (kg water vapor/kg air) and occupancy. Based on the first six attributes, the occupancy is predicted (0 for empty or 1 for occupied). Each entry is consecutively taken every one minute over several days

https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+

Files included:

Data processing files:

- read_data.py

Algorithm files:

- linear_perceptron.py
- rbf_svm.py

Cross-Validation files:

- k_fold_cross_validtion.py

Graph files:

- main_graph.py

For running everything together:

- Main.py

Results:

Preprocessing data:

Currently we can read the data into numpy matrices. In read_data.py, we have three separate functions to read in the data from the three separate data text files (two test sets and one training set). We use pandas, a python data analysis library, to read the information into DataFrames (an object for efficient data manipulation) and then convert it into numpy matrices. The first six columns in the text files are our independent variables: date/time, temperature, humidity, light, CO2, and humidity ratio (which comprises our features matrix) while the last column is our dependent variable is occupancy (which comprises our labels matrix). The only non-numerical feature in our data is the date/time, and we remedy this by disregarding the date and converting the time string into a numerical number of seconds past midnight. We believe that the date does not matter too much because the training data set only encompasses seven different days for over 8,000 data points. We changed all the 0's in the labels matrix to -1 (for unoccupied) so that it would be compatible with perceptron.

Algorithms

Linear perceptron:

We implemented our own linear perceptron algorithm to run on the data. This perceptron algorithm contains similarities with the perceptron algorithm covered in class but with the addition of shuffling. Depending on the order of training data, perceptron can converge to different θ values because the training algorithm stops learning when it stops making mistakes and there can be multiple combinations/permutations of the data in which no mistakes occur (which could mean that it would only learn from a handful of examples). Also, re-permuting the data after each iteration can lead to a faster convergence.

Initially, we ran the perceptron mentioned above on the training data set with all the features to determine if it was linearly separable and determined that the data was not linearly separable. We came to this conclusion by running our perceptron on the training data set for a large number of iterations (10,000 and 100,000). Then used the θ returned to predict the labels from the same data set used to train the perceptron. Theoretically, if this data was linearly separable and the perceptron converged, then it would have made no mistakes when predicting labels from θ in the same data set that it was trained on. However, we found that the algorithm made a non-zero number of mistakes each time it was run, showing that that the data is not linearly separable.

At this point, we decided to test linear perceptron without the time included as a feature because we know that the room the data is representing occupancy of an office room, which would mean that occupancy would cluster during work hours (generally 8 am to 5 pm). This would mean that occupancy of the room would be clustered around specific

times for each day. Also, it is possible that the company could decide to have a party after hours or on the weekend or that a holiday falls on a business day, skewing predictions. We again ran perceptron on the training dataset with all the features but the time. We again determined that it was not linearly separable, in the same manner as before. Therefore showing that even without the date/time included as a feature, it was not linearly separable.

# of iterations	# of mistakes	Date included?
10000	201	Yes
10000	219	Yes
10000	97	No
10000	115	No
100000	143	Yes
100000	136	Yes
100000	95	No
100000	95	No

Now that we know that the data is not linearly separable, we have decided to replace the linear perceptron with a radial basis kernel support vector machine in hopes to separate the data (We had mentioned in our project plan that we would switch the perceptron algorithm out with a non-linear algorithm depending on the linear separability of the data). We will keep our linear support vector machine because we can tune accuracy with the slack variables.

Radial Basis Kernel SVM:

We used sci-kit learn's RBF support vector machine on the training data in hopes to achieve better results than the linear perceptron algorithm. When it came to tuning the parameters for this SVM, we had to decide whether or not to include time because of the reasons mentioned above and determine an appropriate C/slack value. For large C values, the SVM will choose a smaller-margin hyperplane if the hyperplane does a better job of getting all the training points classified correctly. While a small C value will cause the optimizer to look for a larger-margin separating hyperplane, even at the expense of misclassifying more points. To determine an optimal C, we initialized our SVMs with C values of from 0.0001 and 100,000 and counted the number of mistakes it would make on the training set. In addition, we tested the inclusion of the date/time feature and whether or not that would make a significant difference in accuracy (in conjunction with the C values). Our results are as follows:

```
Testing RBF SVM with the time included
+---------+-------------------------------------+
| C value | Mistakes on the training data set |
+---------+-------------------------------------+
|  0.0001 |                1729                 |
|  0.001  |                1729                 |
|   0.01  |                1729                 |
|    1    |                 31                  |
|    10   |                 16                  |
|   100   |                 6                   |
|   1000  |                 1                   |
|  10000  |                 0                   |
|  100000 |                 0                   |
+---------+-------------------------------------+

Testing RBF SVM without the time
+---------+--------------------------+
| C value | Mistakes on training set |
+---------+--------------------------+
|  0.0001 |          1729            |
|  0.001  |          1729            |
|   0.01  |          213             |
|    1    |           72             |
|    10   |           48             |
|   100   |           29             |
|   1000  |           17             |
|  10000  |           11             |
|  100000 |           6              |
+---------+--------------------------+
```

It seems that the time makes a difference in prediction accuracy because the number of mistakes the SVM makes on the training set is higher when time is not included for corresponding C values. With the time included, the SVM improves its accuracy as C values get larger with no significant differences/improvements past C=100. With this in mind, we decided to include time as a feature and a C value of 100 in order to avoid overfitting in our RBF SVM.

Cross Validation

For cross validation, because we determined that the data was not linearly separable, we did not perform k-fold validation on the linear perceptron algorithm. Instead, we did a k-fold cross validation with radial basis kernel on the two testing sets combined. K-fold cross validation works by splitting the data set into k disjoint sets, and training a classifier on k-1 of those and test it on the remaining set. We do this k times, each time holding out a different set as the “testing” part. We then average the performance over all k parts to get an estimate on how well the model will perform in the future on unseen

datasets. In our particular case, we decided to use accuracy percentage as our performance measure. Usually, more folds (a higher k) will lead to better estimates, but each new fold requires training a new classifier, which is computationally expensive. We decided on k=10 folds because it seemed high enough to accurately measure model performance but not too high where it would take an unreasonable amount of time to run.

Our results are as follows when we first ran the 10 fold cross validation on the training data:

```
Would you like to shuffle the data before performing validation? (y, n) n
How many folds? 10
What do you want your C value to be? 100
10 fold cross validation with no shuffling
+--------+----------------------+--------------------+
| Fold # | # of elements in fold | Accuracy Percentage |
+--------+----------------------+--------------------+
|   1    |         1241         |       94.9234      |
|   2    |         1242         |        97.182      |
|   3    |         1242         |       88.3253      |
|   4    |         1241         |       98.9525      |
|   5    |         1242         |       94.0419      |
|   6    |         1242         |       94.0419      |
|   7    |         1241         |       97.0185      |
|   8    |         1242         |       98.3897      |
|   9    |         1242         |       95.6522      |
|   10   |         1242         |       97.9066      |
+--------+----------------------+--------------------+
C value: 100
Average accuracy (mean): 95.6434
Variance of accuracy: 8.707524026
Standard deviation of accuracy: 2.95085140697
```

Overall, the results seemed quite positive as most folds were hitting accuracy percentages in the 90-100 range but there were 1 fold whose accuracy were significantly lower than the others (88%), skewing the overall average accuracy and the standard deviation. We believed that this could have been caused by the fact that in the training set, the data is organized in blocks of unoccupied/occupied. For example, lines 1-16 in the training set are occupied data points (1), then lines 17-830 are all unoccupied (0), then lines 831-1122 are mostly occupied (1). We believed that it would be better to shuffle the data set before cross validating to ensure a more even distribution of unoccupied vs occupied data points in each fold. Our results of the 10 fold cross validation with shuffling are as follows:

```
[Would you like to shuffle the data before performing validation? (y, n) y
[How many folds? 10
[What do you want your C value to be? 100
 10 fold cross validation with shuffling
+--------+----------------------+--------------------+
| Fold # | # of elements in fold | Accuracy Percentage |
+--------+----------------------+--------------------+
|   1    |          1241        |       99.4359      |
|   2    |          1242        |       99.5974      |
|   3    |          1242        |       99.4364      |
|   4    |          1241        |       99.4359      |
|   5    |          1242        |       99.3559      |
|   6    |          1242        |       99.7585      |
|   7    |          1241        |       99.4359      |
|   8    |          1242        |       98.7118      |
|   9    |          1242        |       99.4364      |
|   10   |          1242        |       99.6779      |
+--------+----------------------+--------------------+
C value: 100
Average accuracy (mean): 99.4282
Variance of accuracy: 0.071884542
Standard deviation of accuracy: 0.2681129277
```

As we can see, the results/accuracy of our RBF SVM is far better when shuffling the data before cross validating. From our cross validation, we have determined that our RBF SVM with a C (slack) value of 100 is a good classifier for this data set.

Note: We also decided to run 100 fold cross validation (with shuffling), but it was very time consuming and yielded similar results to our 10 fold cross validation, which is why we decided on k=10 for number of folds.

```
C value: 100
Average accuracy (mean): 99.468632
Variance of accuracy: 0.417577601776
Standard deviation of accuracy: 0.646202446433
```

Graphs

All the data plotted is from the training data set for simplicity's sake (including both test sets would overwhelm the graphs).

Graphs we did not include:

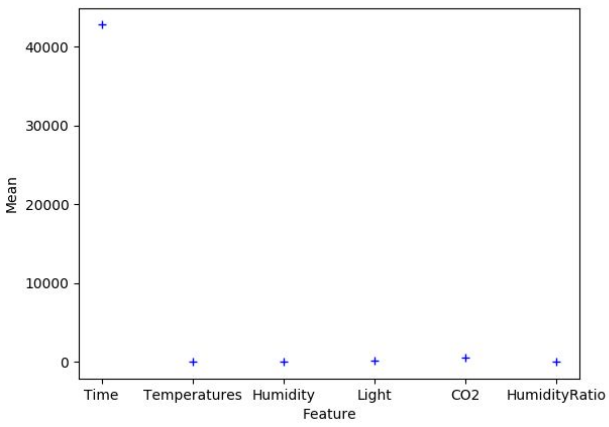
Mean square error: since we already have the accuracy values from the cross validation which we found by comparing the actual occupancy values in the data with the values from the prediction, we do not graph the mean square error.

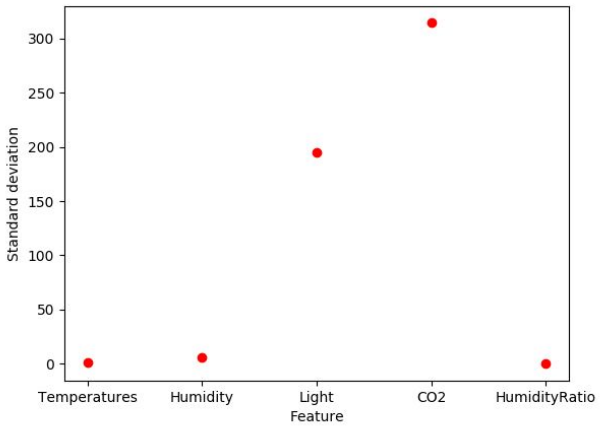
Theta: Theta would be difficult to plot because it would be a 6 dimensional vector. In addition, we did not think it added much to our analysis.

Plots with various sample sizes: we decided not to plot various sample sizes to see if sample size plays a role in predictive accuracy because since we used thousands of samples in our testing, the central limit theorem tells us that a rather large sample size will imply that the distribution of means calculated will approach normality. Therefore, there is no need to plot by sample size as it will not add anything to our analysis. Usually the higher the number of samples means better predictions.

Mean and standard deviation:

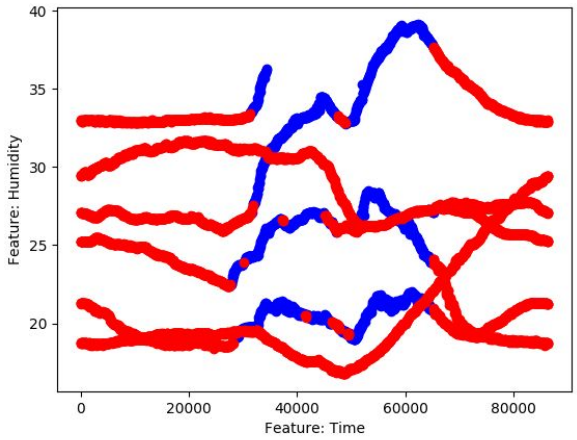
As we stated in the project plan, we will provide graphs of the parameters for each feature, which is the standard deviation and the mean for each feature.

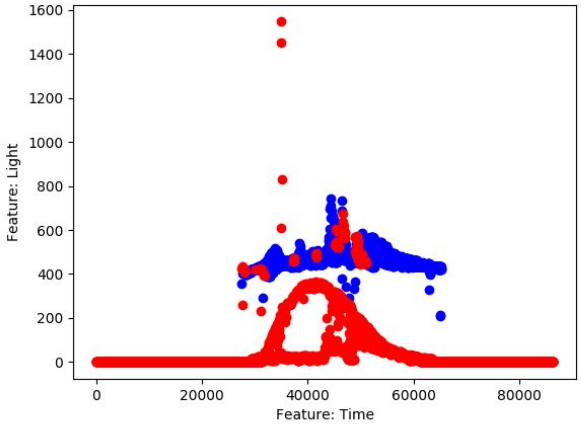


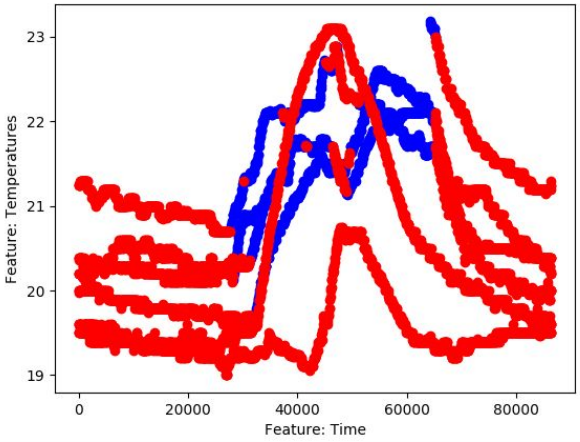


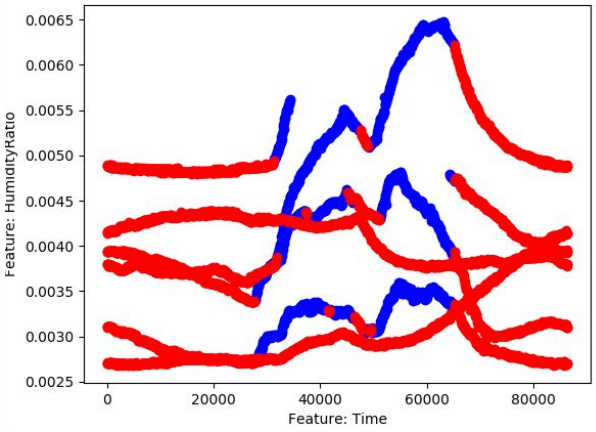
Feature against feature:

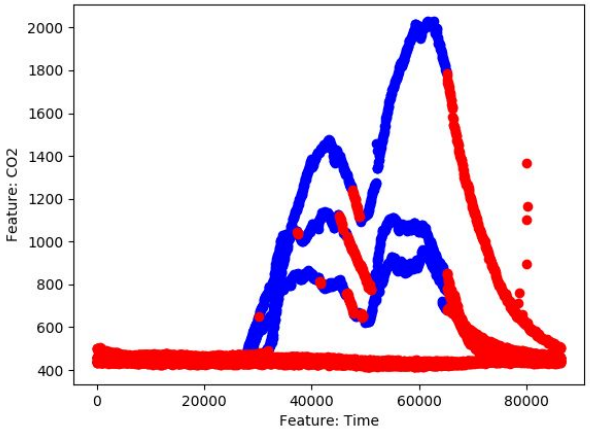
We plotted each feature against the others to visually see the correlation values for each pairing of the 6 features. We included time in this plot because we wanted to see it's pairings with the other features. Note: red represents an occupancy of 1 (occupied) and blue represents an occupancy of -1 (not occupied)(in the original dataset it is denoted as 0).

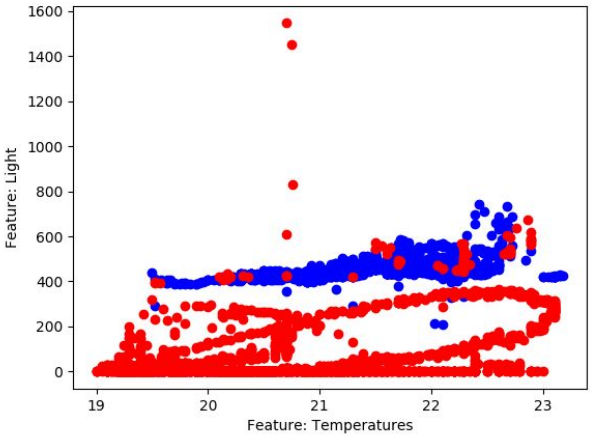

Feature: Humidity
Feature: Time

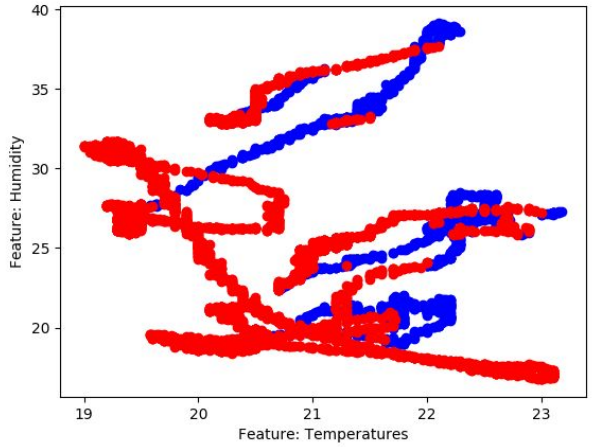

Feature: Light
Feature: Time

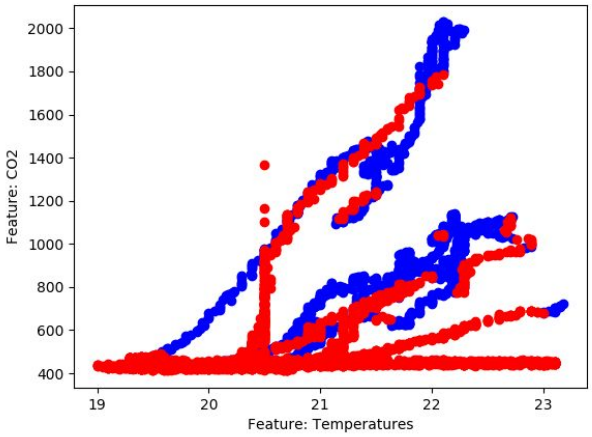

Feature: Temperatures
Feature: Time

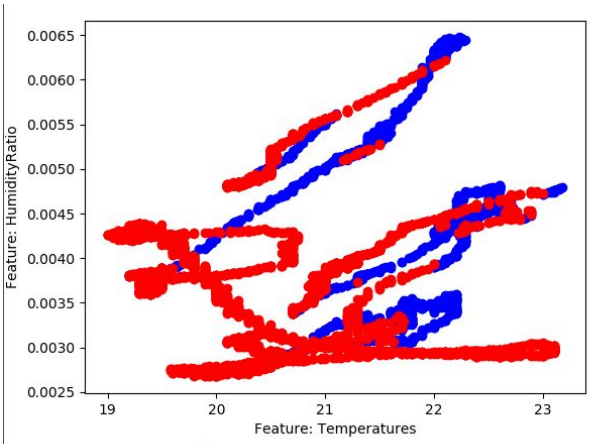

Feature: HumidityRatio
Feature: Time

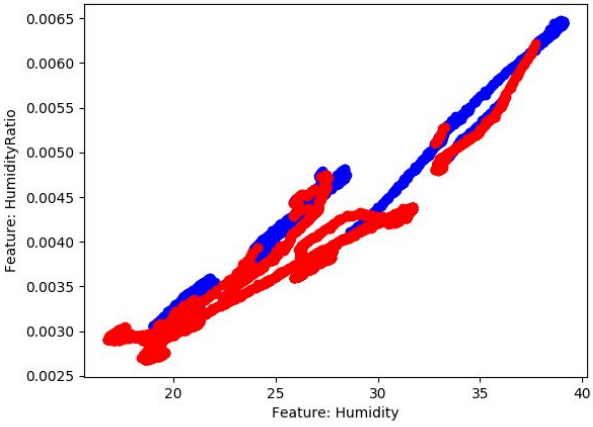

Feature: CO2
Feature: Time

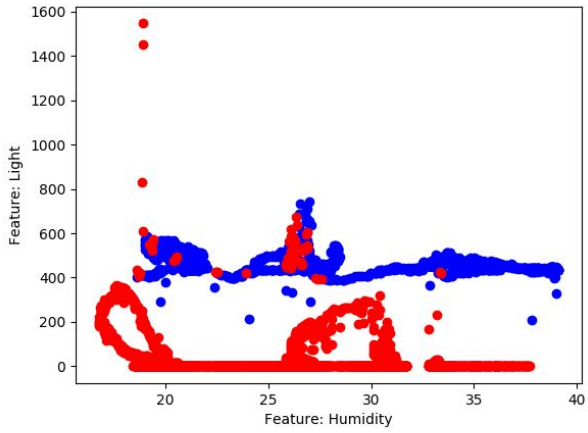

Feature: Light
Feature: Temperatures

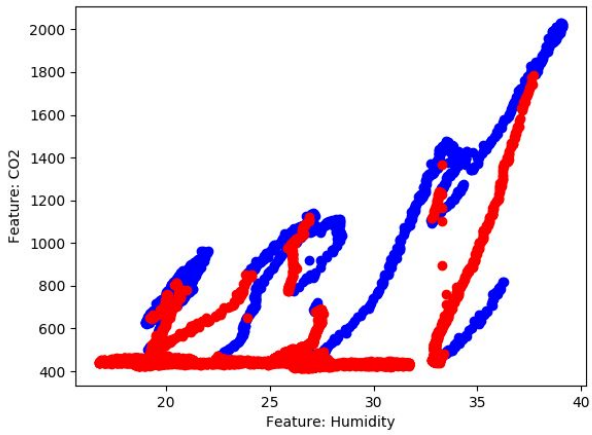

Feature: Humidity
Feature: Temperatures

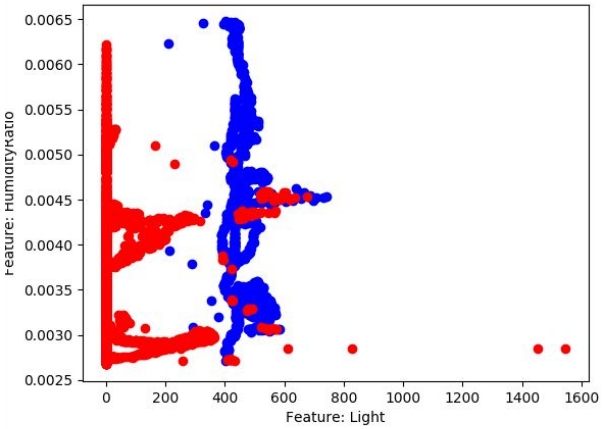

Feature: CO2
Feature: Temperatures

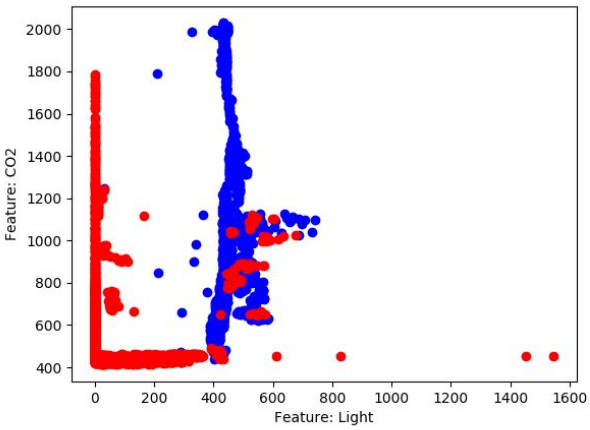

Feature: HumidityRatio
Feature: Temperatures

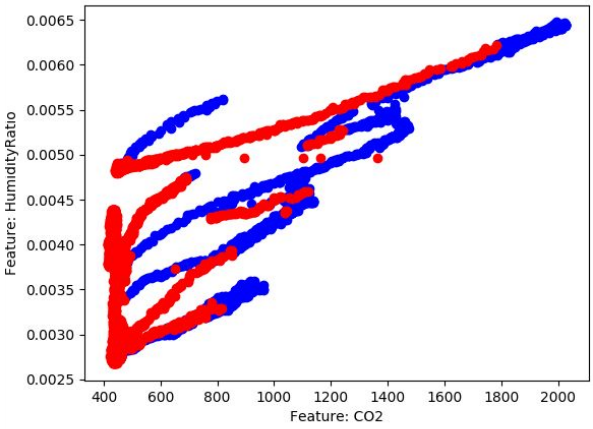

Feature: HumidityRatio
Feature: Humidity


Feature: Light
Feature: Humidity


Feature: CO2
Feature: Humidity


Feature: HumidityRatio
Feature: Light
0.0065
0.0060
0.0055
0.0050
0.0045
0.0040
0.0035
0.0030
0.0025
0
200
400
600
800
1000
1200
1400
1600


Feature: CO2
Feature: Light
2000
1800
1600
1400
1200
1000
800
600
400
0
200
400
600
800
1000
1200
1400
1600


Feature: HumidityRatio
Feature: CO2
0.0065
0.0060
0.0055
0.0050
0.0045
0.0040
0.0035
0.0030
0.0025
400
600
800
1000
1200
1400
1600
1800
2000