

Final project report

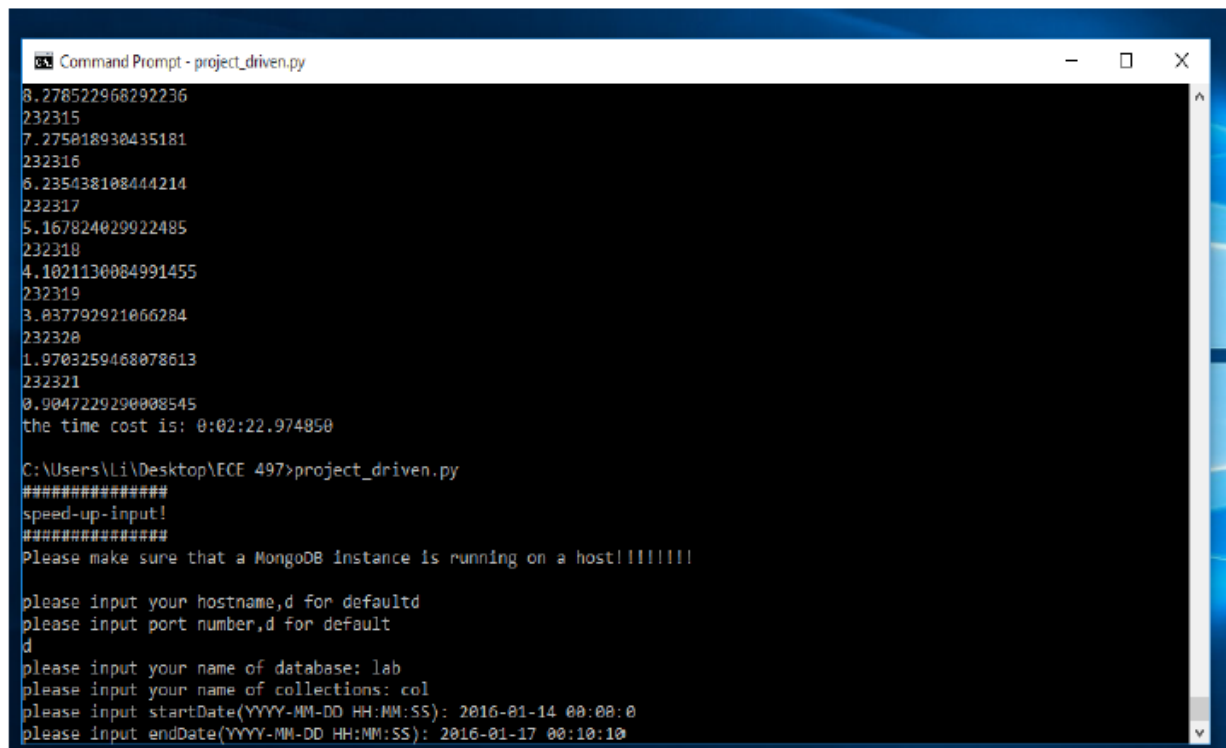
For the past two months, I have finished two projects that have a strong connection with each other.

Project 1: Extract data from database, speeding up fetching speed.

1. Interface

Interface is extremely important in program design, representing a good interaction between end-user and program designer.

Interface itself should be a guider, quickly guide customer towards the start point without waste too much time in preparation stage, collecting correct data as well.



```
8.278522968292236
232315
7.275018930435181
232316
6.235438108444214
232317
5.167824029922485
232318
4.1021130084991455
232319
3.037792921066284
232320
1.9703259468078613
232321
0.9047229290008545
the time cost is: 0:02:22.974850

C:\Users\Li\Desktop\ECE 497>project_driven.py
#####
speed-up-input!
#####
Please make sure that a MongoDB instance is running on a host!!!!!!

please input your hostname,d for defaultd
please input port number,d for default
d
please input your name of database: lab
please input your name of collections: col
please input startDate(YYYY-MM-DD HH:MM:SS): 2016-01-14 00:00:0
please input endDate(YYYY-MM-DD HH:MM:SS): 2016-01-17 00:10:10
```

What I defined in the interface:

- Prompt: instructions to help user avoid regular start-up problem
- Hostname: a valid hostname to be given by user, **localhost as default**
- Port: a correct port number be given by user, in case of “master-slave” multiple pymongo server running in same time.

Those validate a smooth connection.

- database name: user self-define, depend on real setup.
- Collection name: user self-define, depend on real setup.
- Start Date: defined by user
- End Date: defined by user

2. Database connection

Database driven is implemented by Pymongo, a Python distribution containing tools for working with MongoDB.

For this particular case, I used MongoClient to ensure a high quality connection between terminals and database. What it needs are hostname, port, database name and Collection name which has been collected by interface.

All of the relevant code have been encapsulated in database.py

3.Data collecting process

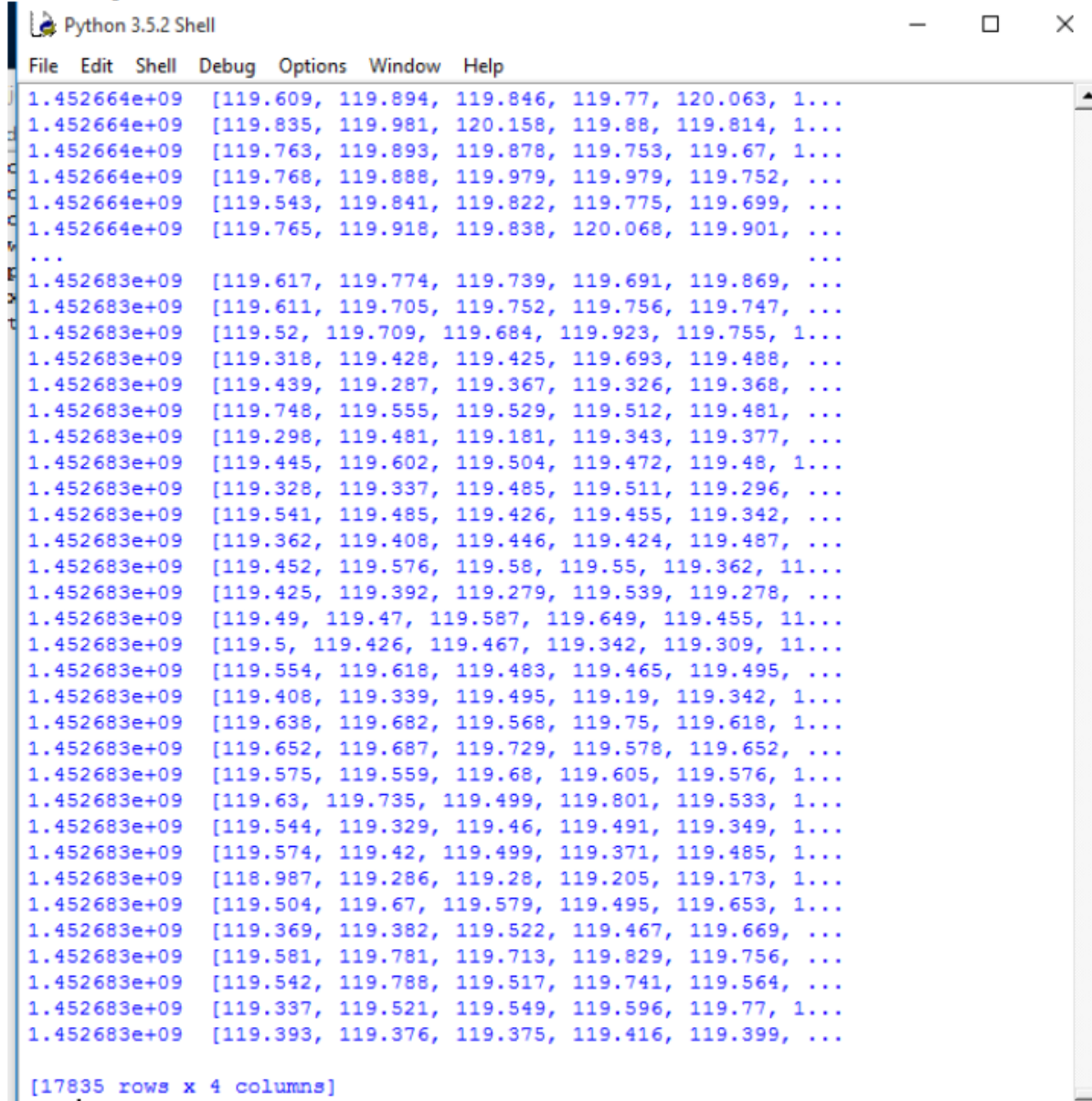
As instructed in the manual, following data should be collected:

- Voltage Being collected by voltage_matrix
- Current Being collected by current_matrix
- Active power Being collected by active power_matrix
- Reactive power Being collected by reactive power_matrix
- The shape is $n \times 4$, being regulated by DataFrames

At this stage I need to point out the reason why I did not follow instructions from the manual. Numpy/List is good for matrix operations, but after you attempt to return values, it is really hard to guarantee the format is $n \times 4$. You may argue that, Numpy.asarray() or Numpy.reshape() could help to solve but it will also change the value of the matrix, which ruins everything. That is the true phenomenon for Numpy array, not to mention list structure, even worser than Numpy. So finally Dataframe is implemented for data storage.

The whole process goes as normal, file input and reading, for loop iteration, data stored inside array, get final array into dataframe.

Result capture as follows:



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
1.452664e+09 [119.609, 119.894, 119.846, 119.77, 120.063, 1...
1.452664e+09 [119.835, 119.981, 120.158, 119.88, 119.814, 1...
1.452664e+09 [119.763, 119.893, 119.878, 119.753, 119.67, 1...
1.452664e+09 [119.768, 119.888, 119.979, 119.979, 119.752, ...
1.452664e+09 [119.543, 119.841, 119.822, 119.775, 119.699, ...
1.452664e+09 [119.765, 119.918, 119.838, 120.068, 119.901, ...
...
1.452683e+09 [119.617, 119.774, 119.739, 119.691, 119.869, ...
1.452683e+09 [119.611, 119.705, 119.752, 119.756, 119.747, ...
1.452683e+09 [119.52, 119.709, 119.684, 119.923, 119.755, 1...
1.452683e+09 [119.318, 119.428, 119.425, 119.693, 119.488, ...
1.452683e+09 [119.439, 119.287, 119.367, 119.326, 119.368, ...
1.452683e+09 [119.748, 119.555, 119.529, 119.512, 119.481, ...
1.452683e+09 [119.298, 119.481, 119.181, 119.343, 119.377, ...
1.452683e+09 [119.445, 119.602, 119.504, 119.472, 119.48, 1...
1.452683e+09 [119.328, 119.337, 119.485, 119.511, 119.296, ...
1.452683e+09 [119.541, 119.485, 119.426, 119.455, 119.342, ...
1.452683e+09 [119.362, 119.408, 119.446, 119.424, 119.487, ...
1.452683e+09 [119.452, 119.576, 119.58, 119.55, 119.362, 11...
1.452683e+09 [119.425, 119.392, 119.279, 119.539, 119.278, ...
1.452683e+09 [119.49, 119.47, 119.587, 119.649, 119.455, 11...
1.452683e+09 [119.5, 119.426, 119.467, 119.342, 119.309, 11...
1.452683e+09 [119.554, 119.618, 119.483, 119.465, 119.495, ...
1.452683e+09 [119.408, 119.339, 119.495, 119.19, 119.342, 1...
1.452683e+09 [119.638, 119.682, 119.568, 119.75, 119.618, 1...
1.452683e+09 [119.652, 119.687, 119.729, 119.578, 119.652, ...
1.452683e+09 [119.575, 119.559, 119.68, 119.605, 119.576, 1...
1.452683e+09 [119.63, 119.735, 119.499, 119.801, 119.533, 1...
1.452683e+09 [119.544, 119.329, 119.46, 119.491, 119.349, 1...
1.452683e+09 [119.574, 119.42, 119.499, 119.371, 119.485, 1...
1.452683e+09 [118.987, 119.286, 119.28, 119.205, 119.173, 1...
1.452683e+09 [119.504, 119.67, 119.579, 119.495, 119.653, 1...
1.452683e+09 [119.369, 119.382, 119.522, 119.467, 119.669, ...
1.452683e+09 [119.581, 119.781, 119.713, 119.829, 119.756, ...
1.452683e+09 [119.542, 119.788, 119.517, 119.741, 119.564, ...
1.452683e+09 [119.337, 119.521, 119.549, 119.596, 119.77, 1...
1.452683e+09 [119.393, 119.376, 119.375, 119.416, 119.399, ...

[17835 rows x 4 columns]
```

4.Data read-in and speed up algorithms

One of the primary goal in this project is, data read-in process should be able to speed up, in terms of bit streams processing rate.

However, how to achieve that? I finally implemented two functions.

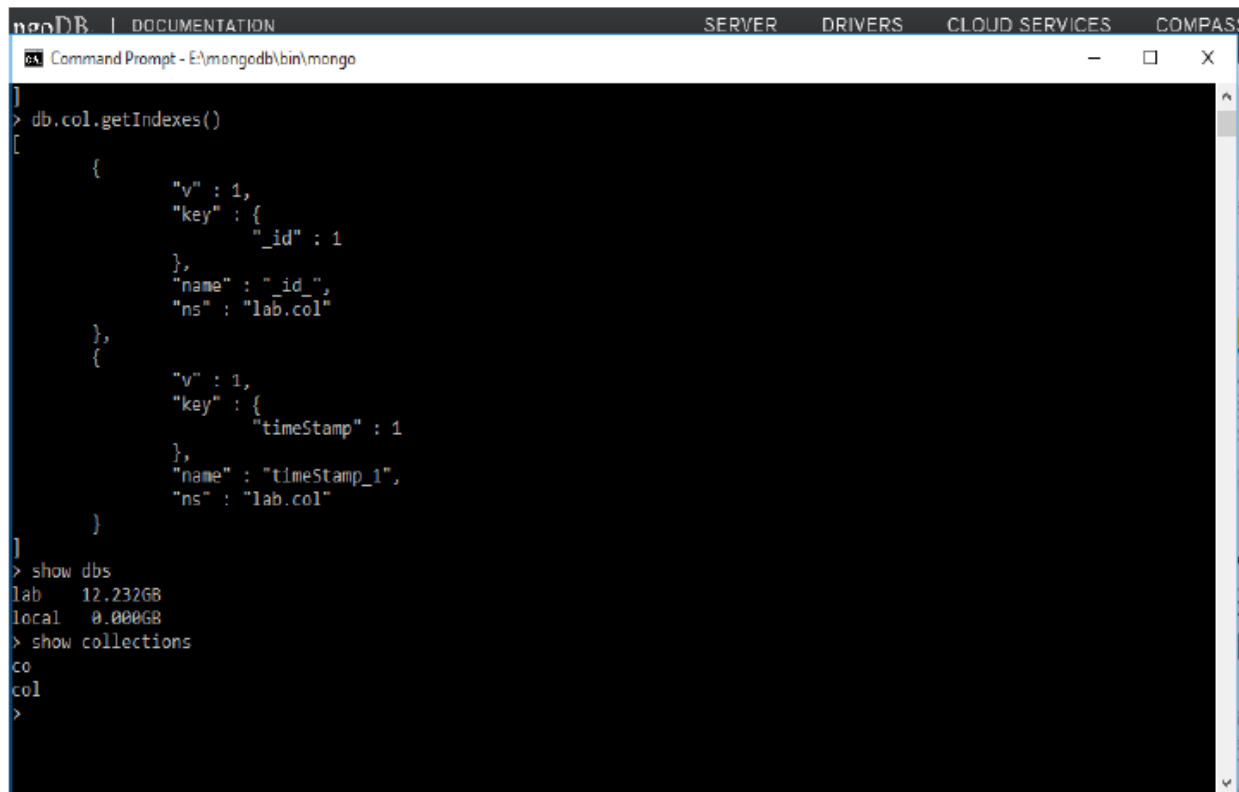
- Create_index function
- Linear approximation model

Create index

An index just like a checkpoint, always matches same or similar results from `find()` command and do their best to search.

Always is the case, a quick-enabled index can save the time to iterate from the head of the database. Index itself serves as a subset of the main database, recording all necessary documents.

Additionally, as you will see later, index only needs to be executed once. After you initialize and run the program once, even then you comment `create_index` code, you can also check the index on MongoDB and find the index still here! When compared with tons of time wasted afterwards as you go through the whole data block, index-based search definitely provides better quality on whole program.

A screenshot of a MongoDB Command Prompt window. The title bar shows 'MongoDB | DOCUMENTATION' and 'SERVER DRIVERS CLOUD SERVICES COMPASS'. The command prompt shows the following commands and output:

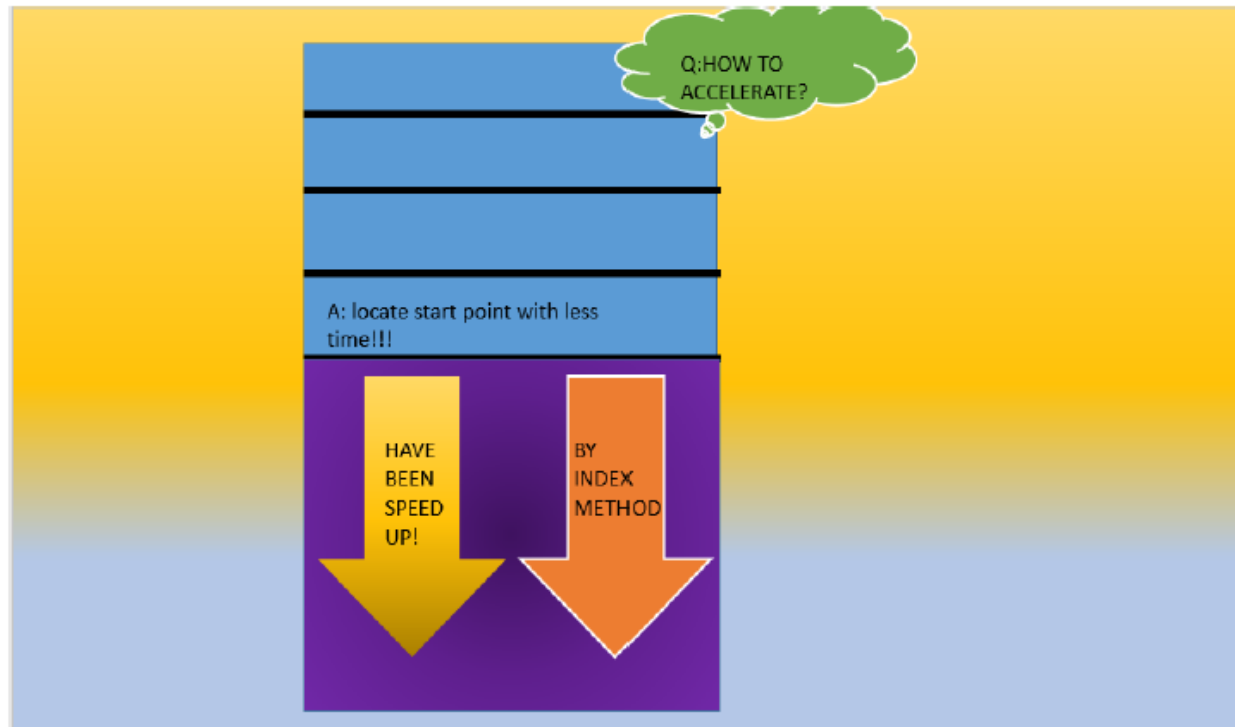
```
> db.col.getIndexes()
[
  {
    "v" : 1,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "lab.col"
  },
  {
    "v" : 1,
    "key" : {
      "timeStamp" : 1
    },
    "name" : "timeStamp_1",
    "ns" : "lab.col"
  }
]
> show dbs
lab 12.232GB
local 0.000GB
> show collections
col
>
```

Index could be established even without command in MongoDB

Linear approximation model

Directly indexing database could be enough to meet the basic requirements. However, when tracing back, it is clear that some other parts can also help to speed up the program performance. And that's exactly where linear approximation model works.

Just consider the whole database as a list. The data after the searching area has been processed with quicker reactions. However, what about the head of the database, in other words, is it possible to use less time to locate the entries we need to find?



So that's where I came up with linear approximation model, based on following empirical experience:

- Date generated with order by nature and the time slots between different entry almost same.
- It is possible to detect start point and end point of the whole database, resulting in better prediction results of implementing linear model without losing too much precision.
- Actually not realistic to introduce half-division method. For the fact that the whole database has 1 million documents with inaccessible containers. So if desired to implement half-division method, a list fully recording the data should be provided prior to find() method, involving too much operations and increasing time complexity.
- However, the design has drawbacks, the most troublesome one is, in order to pursue better performance, the approximation itself needs to make trade-off, which means lower accuracy may involve. But still not too much.

After introducing the self-defined method into the program, I test the data with totally length of three days. I got lower accuracy, nearly 3.7% according to the test, but save 17 seconds totally. I believe maybe it is acceptable, but still it depends on the further purpose of this function and proper revise should be made if possible.

Brief illustrate parameters inside the function:

- Control_const: Maximum allowable units for my laptop's read-in each turns.

- Skip_entry: the number of entry allowed to jump before the index function could reach the goal entry. In order to obtain as much data entries as possible, I set a relative low proportions
- Estimate_const: Give a proportion for the number of entries in all database.
Est= subset data entries/all data entries.
Default value: a day's documents /all documents
- Max_reach : Maximum entries could be skipped. Used to control precision.

5.Data output: DataFrames

Output samples attached as follows:

```
>>> result
```

	current \
1.452762e+09	[0.504933, 0.530315, 0.502148, 0.570384, 0.542...
1.452762e+09	[0.466801, 0.479417, 0.501273, 0.511969, 0.492...
1.452762e+09	[0.516631, 0.49955, 0.524262, 0.488057, 0.5113...
1.452762e+09	[0.509194, 0.515231, 0.508179, 0.544952, 0.505...
1.452762e+09	[0.474017, 0.511906, 0.466689, 0.53887, 0.4705...
1.452762e+09	[0.524982, 0.492858, 0.525734, 0.450039, 0.510...
1.452762e+09	[0.499916, 0.526124, 0.50782, 0.491358, 0.5773...
1.452762e+09	[0.514174, 0.547896, 0.5325, 0.5512, 0.529399,...
1.452762e+09	[0.563062, 0.557886, 0.475574, 0.544144, 0.521...
1.452762e+09	[0.504105, 0.522385, 0.552021, 0.507198, 0.491...
1.452762e+09	[0.489894, 0.500691, 0.514671, 0.510715, 0.586...
1.452762e+09	[0.485818, 0.572955, 0.460141, 0.531472, 0.542...
1.452762e+09	[0.555517, 0.498508, 0.490805, 0.550569, 0.497...
1.452762e+09	[0.53784, 0.530524, 0.572461, 0.510588, 0.5498...
1.452762e+09	[0.600486, 0.526987, 0.487568, 0.557675, 0.480...
1.452762e+09	[0.514808, 0.573531, 0.563896, 0.464416, 0.523...
1.452762e+09	[0.548121, 0.513114, 0.525956, 0.52932, 0.4968...
1.452762e+09	[0.574347, 0.538672, 0.511196, 0.545965, 0.491...
1.452762e+09	[0.57114, 0.544118, 0.543616, 0.570783, 0.5796...
1.452762e+09	[0.519633, 0.538893, 0.477771, 0.532879, 0.499...
1.452762e+09	[0.509774, 0.477367, 0.533487, 0.491428, 0.497...
1.452762e+09	[0.532724, 0.488592, 0.516833, 0.604403, 0.468...
1.452762e+09	[0.556327, 0.494767, 0.551025, 0.513654, 0.490...
1.452762e+09	[0.534854, 0.537329, 0.491672, 0.502069, 0.526...
1.452762e+09	[0.52591, 0.582651, 0.545774, 0.51987, 0.50897...
1.452762e+09	[0.493195, 0.528056, 0.495897, 0.510858, 0.529...
1.452762e+09	[0.45813, 0.534521, 0.48167, 0.473341, 0.50766...
1.452762e+09	[0.48268, 0.506668, 0.466002, 0.512144, 0.5504...
1.452762e+09	[0.513027, 0.504684, 0.535655, 0.530702, 0.520...

```

reactive_power \
1.452762e+09 [33.8886, 36.1024, 34.5239, 36.6496, 35.4749, ...
1.452762e+09 [31.2263, 30.6076, 29.7915, 37.5075, 33.4544, ...
1.452762e+09 [34.7048, 32.3117, 32.2054, 34.4853, 34.1062, ...
1.452762e+09 [37.4234, 35.3514, 32.8938, 34.546, 32.4999, 3...
1.452762e+09 [34.8402, 36.1208, 32.9872, 31.8988, 32.2616, ...
1.452762e+09 [36.5756, 31.2148, 36.0126, 31.8883, 33.3642, ...
1.452762e+09 [39.3475, 35.4523, 32.0023, 31.301, 37.3327, 3...
1.452762e+09 [36.2758, 36.4099, 35.1539, 35.1305, 37.3813, ...
1.452762e+09 [45.1859, 33.9533, 32.852, 35.2428, 37.0651, 3...
1.452762e+09 [34.7761, 35.5189, 34.0747, 33.3008, 32.9824, ...
1.452762e+09 [37.5795, 36.1295, 35.6523, 32.8818, 37.2629, ...
1.452762e+09 [35.8212, 35.8023, 32.8533, 31.6038, 35.4261, ...
1.452762e+09 [42.3728, 31.0755, 31.1142, 32.2008, 32.932, 3...
1.452762e+09 [40.7103, 36.262, 36.0919, 34.2846, 37.1012, 3...
1.452762e+09 [44.9338, 33.0699, 36.3591, 35.3896, 33.0291, ...
1.452762e+09 [35.0641, 38.3506, 39.8013, 31.3061, 38.697, 3...
1.452762e+09 [40.5646, 35.5665, 34.0844, 32.3336, 30.9661, ...
1.452762e+09 [39.3828, 30.8096, 29.7357, 33.8116, 28.4491, ...
1.452762e+09 [38.7791, 35.4613, 35.1327, 34.7053, 35.5477, ...
1.452762e+09 [38.9269, 38.331, 31.6261, 39.237, 32.9184, 33...
1.452762e+09 [33.985, 33.4588, 34.6857, 30.7577, 32.9777, 3...
1.452762e+09 [35.793, 32.0658, 31.6412, 36.7547, 33.6741, 3...
1.452762e+09 [37.6001, 33.8155, 33.3136, 34.9244, 33.7527, ...
1.452762e+09 [36.7171, 32.3664, 34.9075, 33.0758, 34.2331, ...
1.452762e+09 [31.4306, 40.1825, 37.7082, 37.9628, 34.1336, ...
1.452762e+09 [35.5898, 32.3356, 33.2884, 35.8725, 35.4304, ...
1.452762e+09 [29.3343, 32.0453, 33.1478, 30.0129, 32.4071, ...
1.452762e+09 [34.1548, 31.0929, 31.5689, 35.644, 36.4341, 3...
1.452762e+09 [34.9297, 33.3199, 36.8147, 35.168, 38.7808, 3...
1.452762e+09 [38.857, 33.128, 31.2671, 37.4543, 33.5234, 33...
...
1.452838e+09 [203.385, 204.203, 205.747, 205.51, 207.966, 2...
1.452838e+09 [201.325, 205.051, 205.833, 204.528, 203.971, ...
1.452838e+09 [197.67, 204.509, 207.513, 205.68, 203.961, 20...
1.452838e+09 [202.338, 203.28, 204.656, 204.869, 205.302, 2...
1.452838e+09 [198.876, 203.318, 205.909, 204.454, 206.289, ...
1.452838e+09 [199.438, 203.585, 204.41, 204.856, 206.296, 2...
1.452838e+09 [200.847, 201.91, 206.81, 205.254, 205.945, 20...
1.452838e+09 [204.555, 207.526, 205.233, 203.548, 208.851

```

```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help

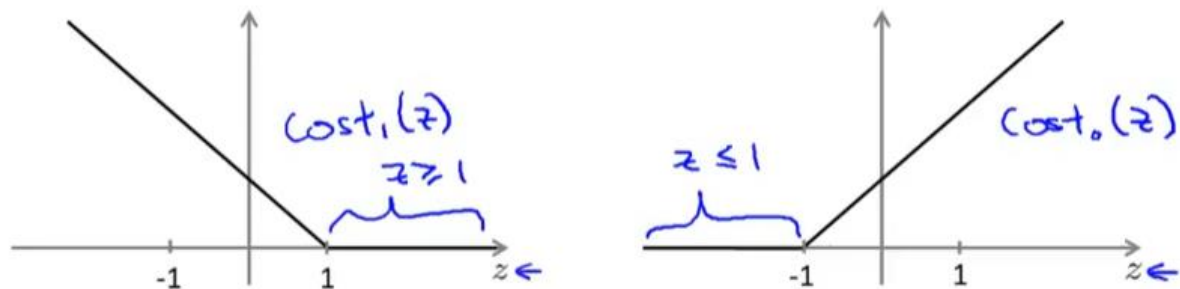
real_power \
1.452762e+09 [50.8603, 53.1471, 50.061, 58.7118, 55.4635, 5...
1.452762e+09 [47.1856, 49.527, 53.1293, 49.5902, 49.487, 53...
1.452762e+09 [52.0169, 51.1609, 54.807, 47.9865, 51.7291, 4...
1.452762e+09 [48.9744, 51.5036, 52.0834, 56.4607, 51.9729, ...
1.452762e+09 [45.4856, 50.3451, 45.8194, 57.0401, 46.8852, ...
1.452762e+09 [52.0512, 50.9554, 52.6603, 44.2016, 52.2482, ...
1.452762e+09 [45.8375, 52.9713, 52.5319, 50.6122, 59.2608, ...
1.452762e+09 [50.5829, 55.6408, 54.2168, 56.981, 52.1946, 5...
1.452762e+09 [51.0291, 58.549, 47.2651, 55.7585, 51.2667, 6...
1.452762e+09 [50.1777, 52.5296, 57.7601, 51.6805, 49.5399, ...
1.452762e+09 [46.004, 48.923, 51.4087, 52.6948, 60.7899, 58...
1.452762e+09 [46.6068, 59.6665, 45.0663, 56.2705, 55.4403, ...
1.452762e+09 [52.4062, 51.9595, 50.8537, 58.7774, 50.6067, ...
1.452762e+09 [50.7767, 53.1054, 59.3914, 51.5823, 55.4117, ...
1.452762e+09 [57.3165, 54.8399, 46.6184, 57.7501, 47.8656, ...
1.452762e+09 [51.5702, 58.1148, 55.6061, 46.7275, 50.1895, ...
1.452762e+09 [52.6514, 51.0774, 53.9888, 55.5235, 51.6849, ...
1.452762e+09 [57.3889, 57.6329, 54.3423, 56.9402, 52.2685, ...
1.452762e+09 [57.442, 55.6402, 55.812, 59.988, 60.7597, 47...
1.452762e+09 [49.383, 52.8761, 48.3371, 51.2307, 50.6327, 5...
1.452762e+09 [51.5778, 47.1881, 54.7207, 51.0464, 50.448, 4...
1.452762e+09 [53.7172, 49.7982, 54.0304, 63.5705, 45.681, 5...
1.452762e+09 [55.9785, 49.4926, 58.0469, 51.58, 48.9563, 49...
1.452762e+09 [53.3219, 56.5675, 48.2212, 51.0573, 53.7934, ...
1.452762e+09 [55.473, 58.235, 54.4124, 50.2162, 51.3079, 50...
1.452762e+09 [48.0315, 55.4074, 50.1069, 50.5543, 53.6755, ...
1.452762e+09 [47.0108, 56.3082, 47.9936, 48.8835, 52.3012, ...
1.452762e+09 [47.4175, 53.0238, 46.8323, 50.892, 56.0449, 5...
1.452762e+09 [51.2604, 51.2501, 53.3843, 53.832, 49.7391, 5...
1.452762e+09 [47.8444, 46.72, 55.7239, 46.9321, 48.876, 49...
...
1.452838e+09 [493.867, 499.891, 486.636, 494.944, 495.305, ...
1.452838e+09 [502.153, 500.849, 493.142, 485.947, 492.134, ...
1.452838e+09 [489.898, 497.219, 491.547, 491.915, 489.368, ...
1.452838e+09 [487.711, 494.944, 498.65, 498.194, 501.146, 4...
1.452838e+09 [492.668, 492.376, 492.695, 496.272, 498.117, ...
1.452838e+09 [491.707, 494.627, 490.835, 502.386, 498.065, ...
1.452838e+09 [493.197, 492.141, 497.348, 492.643, 492.616, ...
Ln: 139968 Col: 0
```

Project 2: Implement support vector regression algorithm (SVR) for extracted data from database to predict total power in the future.

Part 1: Brief illustration on SVR

SVR algorithm is actually an update version from logistic regression. However, it is not realistic to directly employ logistic regression model because it cannot help predict regressively. One way to take care of this issue is, considering partial linearization. After that, the curve will become smooth and not discrete anymore.

$$\rightarrow \min_{\theta} C \sum_{i=1}^m \left[y^{(i)} \underline{\text{cost}_1(\theta^T x^{(i)})} + (1 - y^{(i)}) \underline{\text{cost}_0(\theta^T x^{(i)})} \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$



If $y = 1$, we want $\theta^T x \geq 1$ (not just ≥ 0) $\theta^T x \geq 0$
 If $y = 0$, we want $\theta^T x \leq -1$ (not just < 0) $\theta^T x < 0$

¹SVR algorithm

Additionally, like SVM, SVR algorithm itself is error-resistive, by the way of decision boundary optimization. Always is the case, decision boundary could be a function of features combined as a column vector. However, due to the regulation, the parameter returned by the code may have a range instead of exactly most accurate one. What SVR does is, ignoring the computation results which owns a relatively small values and adjust parameters dynamically. By doing this, it is more likely to obtain better prediction.

Due to the fact that the data set is non-linear (observed from visualization plots), I choose Gauss kernel (RBF model) to make prediction.

The RBF kernel on two samples \mathbf{x} and \mathbf{x}' , represented as feature vectors in some *input space*, is defined as¹

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$

$\|\mathbf{x} - \mathbf{x}'\|^2$ may be recognized as the *squared Euclidean distance* between the two feature vectors. σ is a free parameter. An equivalent, but simpler, definition involves a parameter $\gamma = \frac{1}{2\sigma^2}$:

$$K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$$

² RBF kernel

¹ A note from machine learning tutorial taught by Andrew Ng.

² https://en.wikipedia.org/wiki/Radial_basis_function_kernel

Part 2: Short term load forecast

Methods: How to predict

Something comes first is, after visualization, I find out that data is not a linear shape, suggesting that linear regression cannot perform well as it is a linear method for regression prediction.

For the short load prediction, I particularly adapt Support vector regression(SVR). As mentioned in last report, SVR contains three kernels to fit non-linear situations. For more detail, please check report of last week.

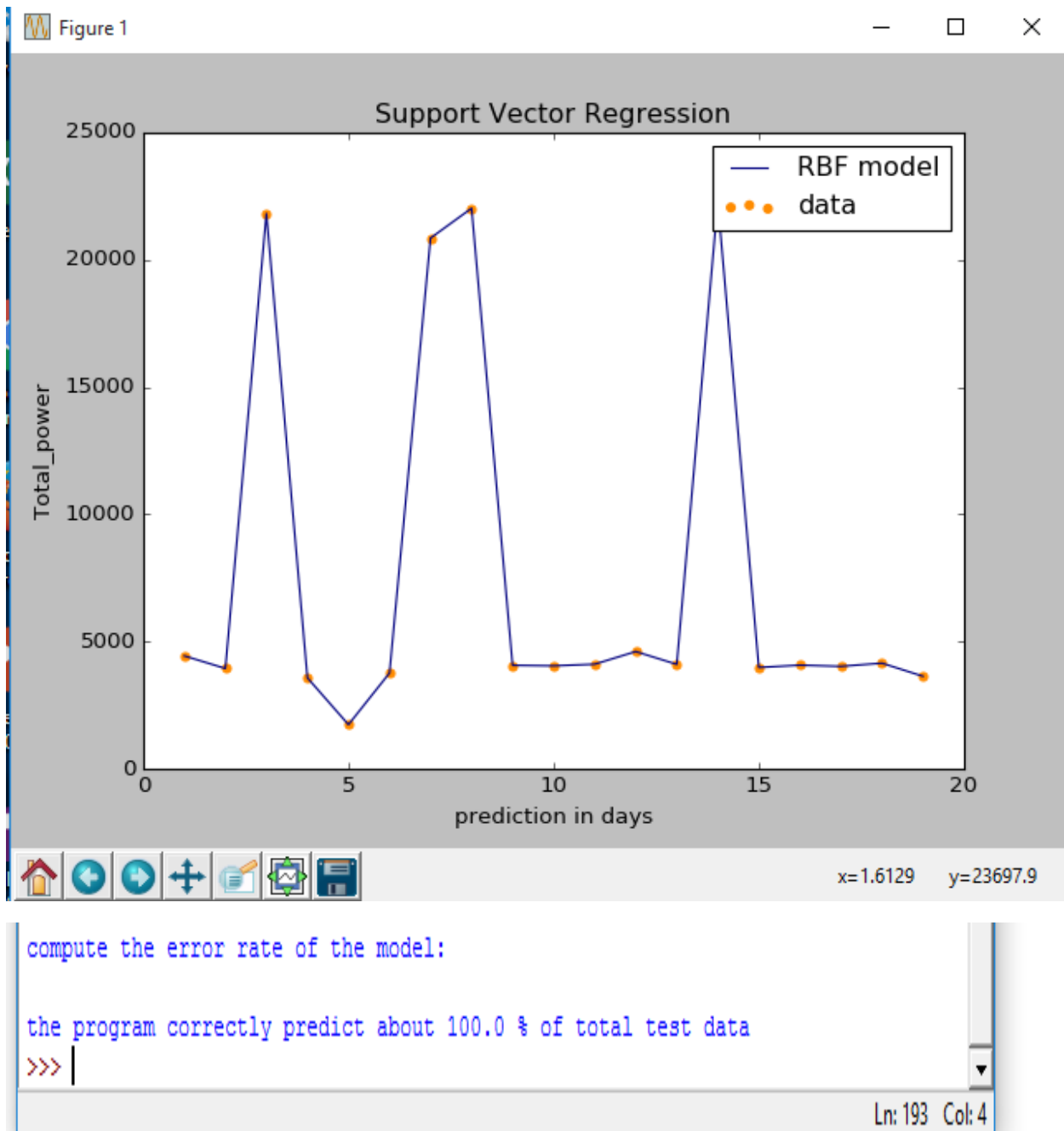
I initially picked up a whole day's total power as input matrix, another day's total power as feather model. I attached results in last week and clearly it is not correct.

After checking the reference material and API, I finally changed my plan. Instead of predicting a whole day, introducing nearly $24 \times 3600 = 78600$ features, which has a huge probability to over fit data, I consider to choose data in same minutes in different days to fit points, which receives a pretty good results

Methods: What is the result

- The size of input data X: $n \times 1$ (n : user defined, should be less than total data length)
- The size of predict matrix y: 1 (1 feather in total)
- Predict_matrix: same length with X

Result capture:



The difference between prediction and test data:

```

///
= RESTART: C:/Users/cli98/Desktop/ECE 497 11_10_2016/ECE 497/point_polish.py =
compute the error rate of the model:
the program correctly predict about 100.0 % of total test data
>>> combine=result_record
array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.])
>>> |

```

I changed another set of parameter:

From the first set of parameters: (see point_polish.py)

```

for i in range(14,32):
    filename='date'+str(i)+'.npy'
    temp=np.load(filename)[0]
    temp1=np.load(filename)[1]
    test=np.hstack((test,temp1))
    combine=np.hstack((combine,temp))
train=combine[0:10]
result_record=train
target=combine[10:11]
predictd=test[0:10]

```

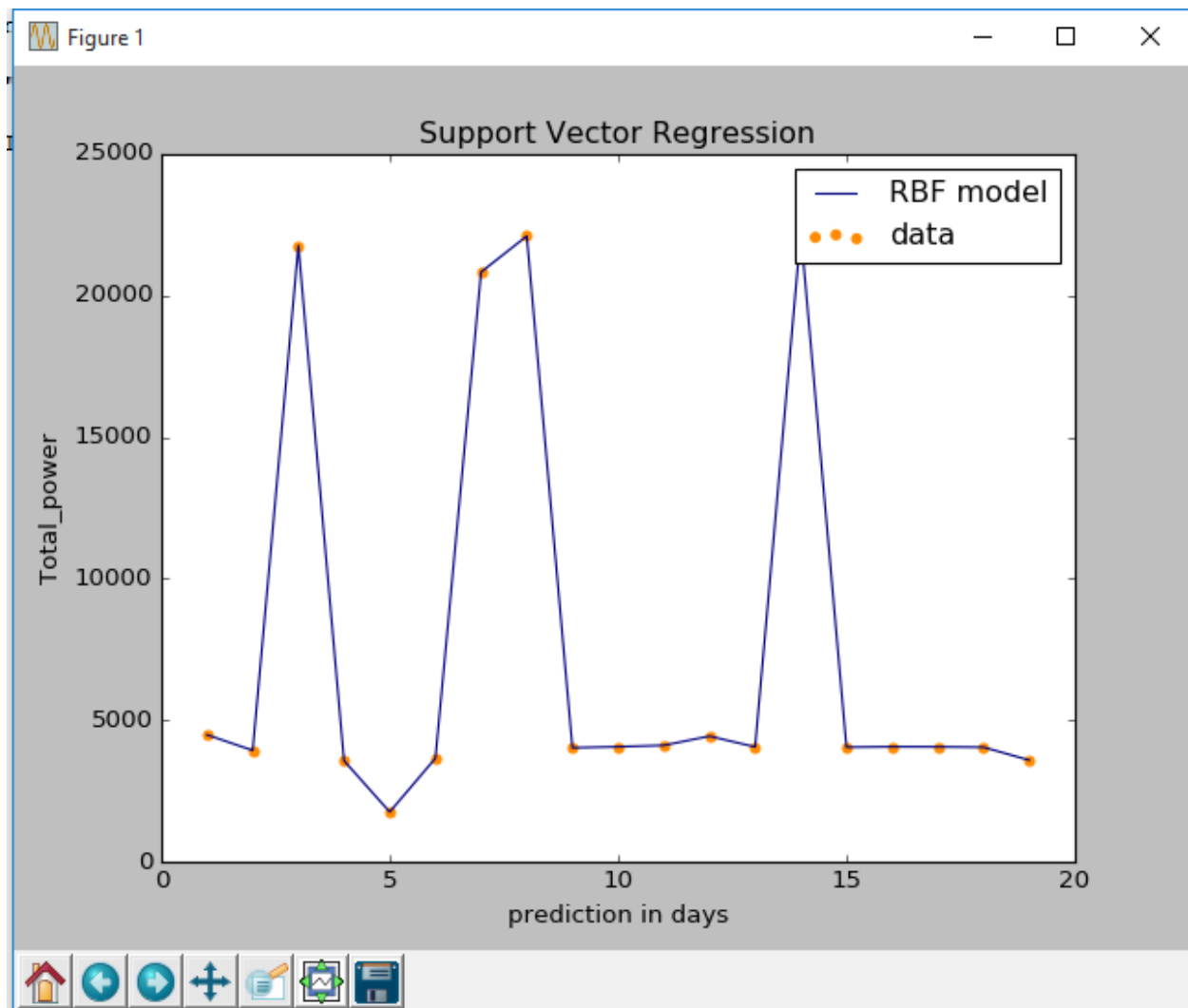
To the following: (see point_polish2.py)

```

start=2
combine=np.load('date13.npy')[start]
test=np.load('date13.npy')[start+1]
for i in range(14,32):
    filename='date'+str(i)+'.npy'
    temp=np.load(filename)[start]
    temp1=np.load(filename)[start+1]
    test=np.hstack((test,temp1))
    combine=np.hstack((combine,temp))
train=combine[0:10]

```

Result capture attached below:



Graph looks same with first one.

I further check the data to see if they are the same.


```

>>> train
array([[ 4434.86838034,   3940.45148335,  21819.34126219,   3571.25908817,
        1732.86818889,   3761.69193538,  20856.55349219,  22029.26726077,
        4062.80823266,   4042.93959768,   4107.22914587,   4605.7415508 ,
        4101.44171728,  22271.15734297,   3979.91172385,   4069.46132818,
        4023.48005752,   4148.04609447,   3635.00281597]])
>>> train.shape
(19,)
>>> train.reshape(1,-1).shape
(1, 19)
>>>
= RESTART: C:/Users/cli98/Desktop/ECE 497 11_10_2016/ECE 497/point_polish.py =
i compute the error rate of the model:
:
i the program correctly predict about 100.0 % of total test data
. >>>
= RESTART: C:/Users/cli98/Desktop/ECE 497 11_10_2016/ECE 497/point_polish.py =
i compute the error rate of the model:
t the program correctly predict about 100.0 % of total test data
w >>> combine-result_record
o array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
e       0.,  0.,  0.,  0.,  0.,  0.])
>>>
= RESTART: C:/Users/cli98/Desktop/ECE 497 11_10_2016/ECE 497/point_polis2.py =
i compute the error rate of the model:
r the program correctly predict about 100.0 % of total test data
>>> train
o array([ 4482.31708846,   3932.94679963,  21761.3202446 ,   3566.60330641,
r       1757.65580871,   3654.87304334,  20832.38544435,  22103.39288354,
o       4025.18429088,   4061.43942908,   4111.14931132,   4435.48994097,
s       4056.04870375,  22247.07025108,   4045.30033631,   4062.07919332,
l       4058.63039944,   4044.16997128,   3587.74894346])
o >>> combine-result_record
array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
       0.,  0.,  0.,  0.,  0.,  0.])
>>> |

```

Ln: 94 Col: 4

Dataset in different days are similar but not same.

Test result:

```

Python 3.5.1 Shell
File Edit Shell Debug Options Window Help
Python 3.5.1 (v3.5.1:37a07cee5969, Dec 6 2015, 01:54:25) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\cli98\Desktop\ECE 497 11_10_2016\ECE 497\point_polish.py =
i compute the error rate of the model:
t the program correctly predict about 100.0 % of total test data
>>> train
array([[ 4434.86838034,   3940.45148335,  21819.34126219,   3571.25908817,
        1732.86818889,   3761.69193538,  20856.55349219,  22029.26726077,
        4062.80823266,   4042.93959768,   4107.22914587,   4605.7415508 ,
        4101.44171728,  22271.15734297,   3979.91172385,   4069.46132818,
        4023.48005752,   4148.04609447,   3635.00281597]])
>>>

```

Reference

1. Machine learning, Stanford university, Coursera, Professor Andrew Ng.
2. What is SVR? URL: https://en.wikipedia.org/wiki/Radial_basis_function_kernel