# Table of Content

## 1. Introduction

This paper conducts research in aspect of data processing and visualization. We are using machine learning algorithms to explore the structure and relations of provided data and trying to find out meanful data models for decision making. Prior to the paper, a raw database consisting of ammeter records is provided. These records include certain key indices (model, location, power, voltage, temperature, etc) of an ammeter.

The objective of this paper is to determine whether an ammeter functions normally or not, based on the history records and the learned data model. Section 2 lays out how the raw database is migrated to and processed with a more suitable tool - mongodb. Section 3 focused on how the machine learning algorithm, specifically PCA (Principal component analysis) help us to calculate the proper data model for us.

## 2. Data Preparation

As the key indices of an ammeter are scattered across tables in the raw database, our first goal is to gather the data together and output the result in the form that the ammeter ID is used as the primary key and the rest properties (model, location, power, voltage, temperature and etc.) are the values to the primary key. The following is the structure of the raw database:

**Database.mdb**

```
// Structure of table 'Load' in Microsoft Access
---------------------------------------------------------------------------
ID     | IntTime         | Meter        | Town     | IntVal | Channel
-------|-----------------|--------------|----------|--------|-----------------
266950 | 7/6/2013 10:30 | 260001490_D | MAYWOOD | 237.9  | Ch-3:Average ...
266951 | 7/6/2013 10:30 | 260001490_D | MAYWOOD | 2.25   | Ch-1:KWh ...

// Table 'Load2' is the same with 'Load'

// Structure of table 'Temp' in Microsoft Access
-----------------------------------------------------------------------------
ID| AMIModel | Meter        | Town   | Date            | T    | Latitude | ...
--|----------|--------------|--------|-----------------|---------------------
1 | I210+    | 239827814_G | BERWYN | 11/2/2014 23:44 | 71.6 | ...      | ...
2 | I210+    | 239827814_G | BERWYN | 11/2/2014 19:20 | 71.6 | ...      | ...
```

As we can see that the power (for the `Channel` field, the value of which starts with `Ch-1: KWh` in table `Load` ), voltage (for the `Channel` field, the value of which starts with `Ch-3: Average voltage` in table `Load` ), and temperature (the value of `T` field in table `Temp` ) are scattered, for the purpose of furthe process, we need to merge the three tables( `Load` , `Load2` and `Temp` ) and produces results having the following structure:

**Merged_Output**

```
// the final structure from mongodb
{
    "_id" : 1,
    "town" : "BELLWOOD",
    "amiModel" : "FocusAX",
    "year" : 2013,
    "meter" : "260002646_D",
    "month" : 11,
    "day" : 23,
    "latitude" : 41.8815383677918,
    "longitude" : -87.8819471824789,
    "t0" : 48,
    "t1" : 46.5,
    "t2" : 45,
    ...
    "t11" : 30.2,
    "k0" : 0.2097,
    "k1" : 0.2883,
    ...
    "k11" : 0.1072,
    "v0" : 245,
    "v1" : 244,
    "v2" : 244,
    ...
    "v11" : 244
}
```

In the final structure, 12 temperature values (t0-t11), 12 power values (k0-k11)and 12 voltage values (v0-v11) of an ammeter at a given date are all presented. In other words, for one ammeter, the temperature, power and voltage are sampled every 2 hours. (The fact is that not all 12 temperature, power and voltage vlaues are sampled or recorded for each ammeter in the raw database, hence value insertion needs to be done at a later stage).

## 2.1 Migration

When say data migration, we means transferring data from one database to another. Microsoft Access is resource-demanding and does not run well on the computer we use. Limited by computer resource (both software and hardware), we came to the idea of using another database-Mongo DB to replace Microsoft Access. Mongo DB can be easily deployed in a distributed structure (which means certain computation intensive job can be completed through the coorperation of multiple hosts).

> MongoDB (from humongous) is a cross-platform document-oriented database. Classified as a NoSQL database, MongoDB eschews the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas (MongoDB calls the format BSON), making the integration of data in certain types of applications easier and faster. Released under a combination of the GNU Affero General Public License and the Apache License, MongoDB is free and open-source software. [1]

Here, we are using `odbc` [2] driver[3] to read the *.mdb file (Microsoft Access file format) and using mongodb driver[4] to write the data into the Mongo DB.

```
// data_migrate.py
# local Microsoft Access database file
localDBStr = 'DRIVER={Microsoft Access Driver (*.mdb)};DBQ=Database.mdb'
localConn = pyodbc.connect(localDBStr)
cursor = localConn.cursor()

# remote MongoDb server
remoteDBUrl = 'mongodb://<user>:<password>@<hostname>:27017/am'
remoteClient = MongoClient(remoteDBUrl)
remoteDB = remoteClient.am
```

Most of the fields in Microsoft Access can be seamlessly migrated to Mongo DB. However, the primary keys in the two database are different, as Microsoft Access uses name `ID` and Mongo DB uses name `_id`. Both `ID` and `_id` server as the same purpose.

### 2.2 Aggregation

#### 2.2.1 Joining and Separation `Load` collections

Before aggregation, the following job needs to be done

- First, as the collection (corresponding to the concept of table in SQL database) `Load` and `Load2` have the same structure, it is necessary to merge thest two first. ( `Load` , `Load2` => `Load` )
- Second, as the power and voltage records are entangled in one collection, we need to seperate them from each other into different collections ( `Load` => `Kwh` and `Voltage` )

The first job can be done through the following script (running in Mongo DB server shell):

```
// Load.js
db.Load2.find().forEach(function(doc) {
  // the final script may vary for performance consideration
  // db.Load.insert(doc);
});
```

For the second job, we are using the value of `channel` field to seperate the power and voltage records.

```
// kwh.js
// voltage.js

// Load
---------------------------------------------------------------------------
ID      | IntTime         | Meter        | Town     | IntVal | Channel
-------|----------------|-------------|---------|--------|-----------------
266950 | 7/6/2013 10:30 | 260001490_D | MAYWOOD | 237.9  | Ch-3:Average ...
266951 | 7/6/2013 10:30 | 260001490_D | MAYWOOD | 2.25   | Ch-1:KWh ...

//   ||
//   ||
// \\  //
//  \\//
//   \/

// Kwh
-----------------------------------------
ID | Timestamp       | Meter        | Kwh
---|----------------|-------------|------
1  | 7/6/2013 10:30 | 260001490_D | 2.25

//  Voltage
-------------------------------------------
ID | Timestamp       | Meter        | Voltage
---|----------------|-------------|--------
1  | 7/6/2013 10:30 | 260001490_D | 237.9
```

### 2.2.2 Aggregating colletions

We enter into one of the most important and usefule features of Mongo Db - Aggregation. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result.
When grouping the records, we use date value as the group keys, and put all the records in the corresponded bins. As the result shows, asssociated with a given date, there is a list of power, voltage or temperature.

```
Kwh  => Kwh_Grouped
Voltage  => Voltage_Grouped
Temp  => Temp_Grouped
```

```
// commone aggregation pipeline
db.CollectionName.aggregate([{
  $group: { }
}, {
  $project: { }
}, {
    $out: 'Output Collection Name'
}], {
  allowDiskUse: true,
});

// kwh_grouped.js
// Kwh_Grouped
------------------------------------------------------------------------------
_id                   | Meter       | Year | Month | Day | List
----------------------|-------------|------|-------|-----|-------------------
260001490_D2013-01-01 | 260001490_D | 2013 | 1     | 1   | [<Timestamp, Kwh>]

// voltage_grouped.js
// Voltage_Grouped
------------------------------------------------------------------------------
_id                   | Meter       | Year | Month | Day | List
----------------------|-------------|------|-------|-----|-------------------
260001490_D2013-01-01 | 260001490_D | 2013 | 1     | 1   | [<Timestamp, Voltage>]

// temp_grouped.js
// Temp_Grouped
----------------------------------------------------------------------------
_id                   | Meter       | Year | Month | Day | List
----------------------|-------------|------|-------|-----|-----------------
260001490_D2013-01-01 | 260001490_D | 2013 | 1     | 1   | [<Timestamp, T>]
```

Note:
The value of `_id` field in the above three grouped collections is the concatenation result of Meter and the Date with only year, month, and day. This boosts the performance in the next stage when we try to merge the grouped collections.

### 2.2.3 Merging grouped colletions

We have aggregated and acquired the grouped data (by meter and date). The following script helps us to merge the three grouped collections into one:
`Kwh_Grouped` , `Voltage_Grouped` and `Temp_Grouped` => `Merged`

```
// final script may vary for performance consideration
// merged.js
// Merged
db.Temp_Grouped.find().forEach(function(tDoc) {
  var _id = tDoc._id;
  // _id is the concatenation result of Meter and the Date with only year,
  // month, and day
  var kwhDoc = db.Kwh_Grouped.findOne({ _id: _id });
  var vDoc = db.Voltage_Grouped.findOne({ _id: _id });

  // producing a new document
  var newDocument = {};
  // inserting the new document into a new collection
  db.Merged.insert(newDocument);
});
```

### 2.2.4 Inserting values

As mentioned before, the database do not have all the expected records (12 pieces of data for each index should be recorded) for data modeling. At the final stage of data preparation, we employ the simple idea of value insertion to output a relatively integral collection.The idea works as follows:

```
# merged_output.py

# 1. Allocating an empty list of the size of 12
t_list = [0 for i in range(0, 12)]
for record in result['TList']:
  timestamp = record['Timestamp']
  hour = timestamp.hour
  # 2. put the value into the empty list
  t_list[hour / 2] = record['T']

# 2. Iterating the filled list. For a zero value at a certain index, if the
# index is 0 (first item in the list), assigining the nearmost non-zero value
# to the zero value; if the index is between the first and last ones,
# assigning the previous value to the index if the value of next index is 0 or
# assigning the average of the previous and next ones to the index; if the
# index is the last one, assigning the value of previous index to the last one.
def insert(dataset):
  dataset_variant = [0 for i in range(0, len(dataset))]
  #print 'dataset        ', dataset
  for index, data in enumerate(dataset):
    if data == 0:
      # when index is 0, first one.
      if index == 0:
        for data1 in dataset:
          if data1 != 0:
            dataset_variant[0] = data1
            break
      # when index is between [1, 10].
      elif index < len(dataset) - 1:
        # [43, 0, 0], index-1 => 43, index => 0, index+1 => 0.
        if dataset[index + 1] == 0:
          dataset_variant[index] = dataset_variant[index - 1]
        # [43, 0, 45], index-1 => 43, index => 0, index+1 => 45.
        else:
          dataset_variant[index] = (dataset_variant[index - 1] + dataset[index + 1]) / 2
      # when index is 11, last one.
      else:
        dataset_variant[index] = dataset_variant[index - 1]
    else:
      dataset_variant[index] = data
  return dataset_variant
```

After insertion, a collection named `Merged_Output` is produced. With the data in `Merged_Output`, we can begin to study the data model.

## 2.3 Export

We are using an export tool provided by Mongo DB to do the heavylifting. The output result is a cvs file.

```
mongoexport --host <hostname> --db <database> --collection <collection name> --csv --out
<filename> --fields <field names> --username <username> --password <password> -q <query json
string> --limit <document number>
```

We can export the document in collection `Merged_output` , and output whatever numbers of documents as we needs by only chaning the `--limit` parameter.

## 3. Data Anaysis

We are using IPython notebook service and scikit-learn machine learning algorithm package for the calculation and data visualization.

### 3.1 PCA introduction

Principal Component Analysis, or simply PCA, is a statistical procedure concerned with elucidating the covariance structure of a set of variables. In particular it allows us to identify the principal directions in which the data varies. PCA is a common technique for finding patterns in data of high dimension.

### 3.2 Computing the Principal Components

#### 3.2.1 Preprocess Dataset

From section 2, we can easily get the raw ammeter data according to the features we need to analyze. For example, a $12 * n$ matrix can be extracted from the exported cvs file, wherein power values at 12 time points in a single day form a row and there are $n$ such rows ($i = n, j = 12$):

$$\begin{pmatrix} k_{00} & k_{01} & \dots & k_{0j-1} & k_{0j} \\ k_{10} & k_{11} & \dots & k_{1j-1} & k_{1j} \\ \dots \\ k_{i0} & k_{i1} & \dots & k_{ij-1} & k_{ij} \end{pmatrix}$$

For PCA to work properly, we subtract the mean from each of the data dimensions. The mean substracted is the average accross each dimensions. So all the $k_x$ values have $\bar{k}_x$ (the mean of $k_x$ values of all the data points) subtracted. This produces a data set whose mean is zero.

$$\begin{pmatrix} k_{00}-\bar{k}_0 & k_{01}-\bar{k}_0 & \dots & k_{0j-1}-\bar{k}_0 & k_{0j}-\bar{k}_0 \\ k_{10}-\bar{k}_1 & k_{11}-\bar{k}_1 & \dots & k_{1j-1}-\bar{k}_1 & k_{1j}-\bar{k}_1 \\ \dots \\ k_{i0}-\bar{k}_i & k_{i1}-\bar{k}_i & \dots & k_{ij-1}-\bar{k}_i & k_{ij}-\bar{k}_i \end{pmatrix}$$

#### 3.2.2 Calculate the covariance matrix

First, let us review the formula for covariance:

$$cov(X,Y) = \frac{\sum_{i=1}^{n}(X_i-\bar{X})(Y_i-\bar{Y})}{(n-1)}$$

The formula for calculating the covariance matrix of a set of data with $n$ dimensions is:

$$C^{n\times n} = (c_{i,j}, c_{i,j} = cov(Dim_i, Dim_j)),$$

where $C^{n\times n}$ is a matrix with $n$ rows and $n$ columns, and $Dim_x$ is the $x$th dimension.

### 3.2.3 Calculate the eigenvectors and eigenvalues of the covariance matrix

In computational terms, the principal components are found by calculating the eigenvectors and eigenvalues of the covariance matrix. This process is equivalent to finding the axis system in which the covariance matrix is diagonal. The eigenvector with the largest eigenvalue is the directions of greated variation, the one with the second largest eigenvalue is the (orthogonal) direction with the next highest variation and so on.
The eigenvalues of $C$ are defined as the roots of:

$$determinant(C - \lambda I) = |(C - \lambda I)| = 0$$

where $I$ is the $n \times n$ identity matrix. The equation is called the characteristic equation and has $n$ roots. Let $\lambda$ be an eigen value of $C$, then there exists a vector $x$ such that:

$$Cx = \lambda x$$

### 3.2.4 Choosing components and forming a feature vector

As the eigenvectors are found from the covariance matrix above, we then order them by eigenvalues from highest to lowest and we can see the components in order of significance and then we can ignore the components of lesser significance. We do lose some information, but as the eigenvalues are small, we don't lose much. Specifically, if we have $n$ dimensions in the dataset, $n$ eigenvectors and eigenvalues can be acquired, and we choose only the first $p$ eigenvectors and the final data set has only $p$ dimensions.
Now, we need to form a *feature vector*. It is constructed by taking the eigenvectors we keep previously and forming a matrix with thest eigenvectors in the columns.

$$FeatureVector = (eig_1, eig_2 \dots, eig_n)$$

### 3.2.5 Deriving the new dataset

The final step of PCA is to derive the new dataset with the chosen components (eigenvectors). Taking the transpose of the vector and multiply it on the left of the original dataset transposed:

$$FinalData = RowFeatureVector \times RowDataAdjust$$

## 3.3 Application of PCA

Luckily, most math packages provide implementations of PCA algorithm, we are using `scikit-learn` python package for the computation. For the comparison purpose, we exported serval csv files of different sizes, which are 1000, 10000 and all.
The images only show the results of the training data of the size of 1000 and 10000. In each of the following images, there are four plots, with the 1st and 2nd ones plotting the the eigenvalues and transformed training data of 1000 samples and the 3rd and 4th one plotting the the eigenvalues and transformed training data of 10000 samples.

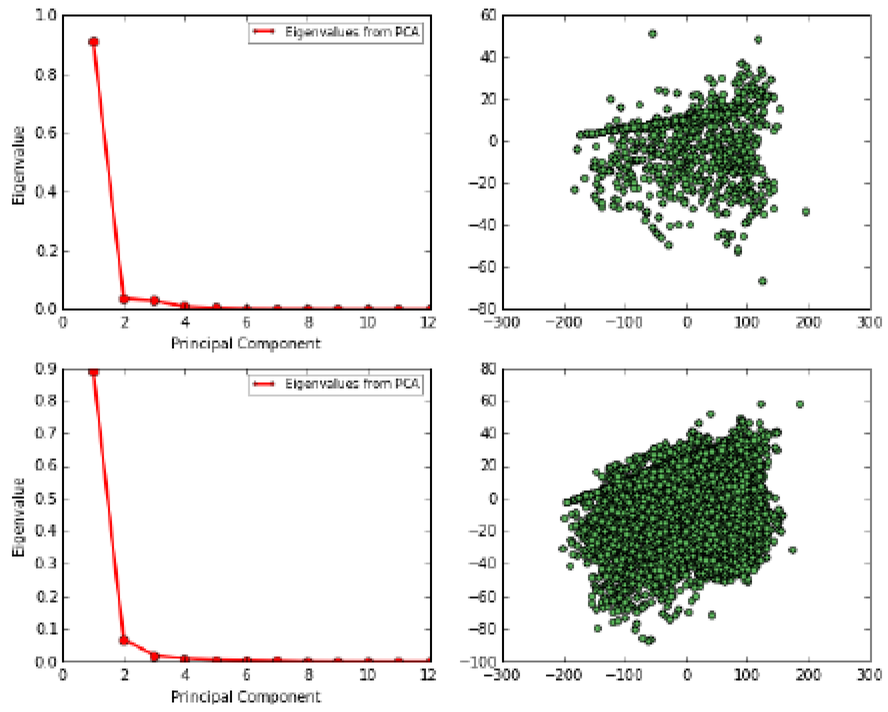
```
from sklearn.decomposition import PCA
# d is the input training dataset, n is number of components to keep
def do_pca(d, n):
    pca = PCA(n_components=n)
    X = pca.fit_transform(d)
    print 'pca.explained_variance_ratio_', pca.explained_variance_ratio_
    return X
```

In the above code, `n` indicates the number of components (features) we keep, as mentions before, the first two components take the lion's share of all the feature.(See the each plot, after the second plot, the trend drops steeply, and the rest componnets only take a small part in the weight.). Thus we use the first two components for the analysis.
At first, we feed the algorithm with each dataset of temperature, power and voltage seperately:

### 3.3.1 Temperature Plot

```
# temperture - t_fields t0, t1, ... t1
tx = raw_df[t_fields].values
tx_pca, t_eigvals = do_pca(tx, 2)
# draw
```

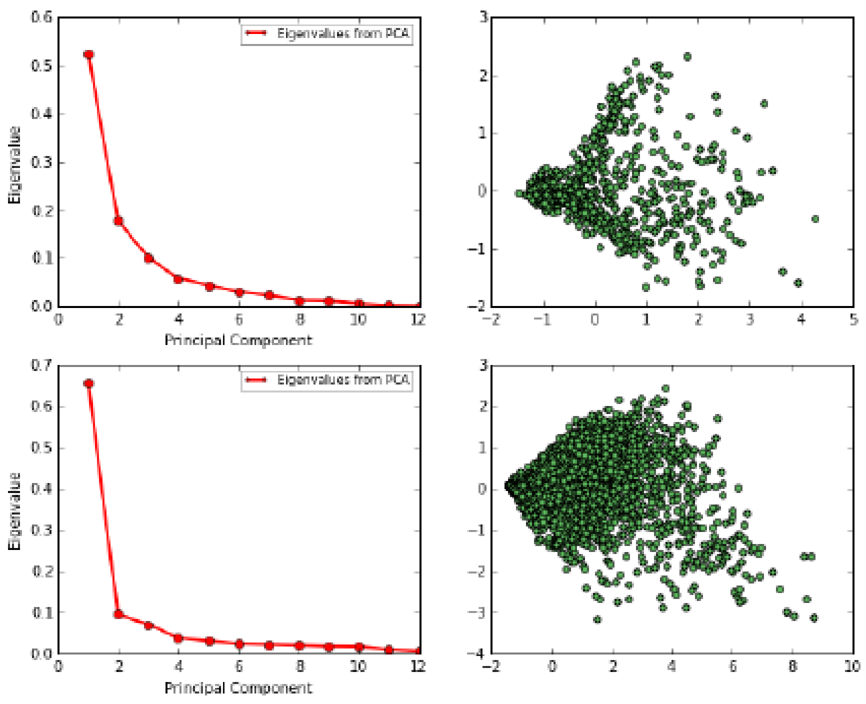
Fig 3.1 PCA - Temperature



### 3.3.2 Power Plot

```
# k_fields - kwh k0, k1, ... k11
kx = raw_df[k_fields].values
kx_pca, k_eigvals = do_pca(kx, 2)
# draw
```

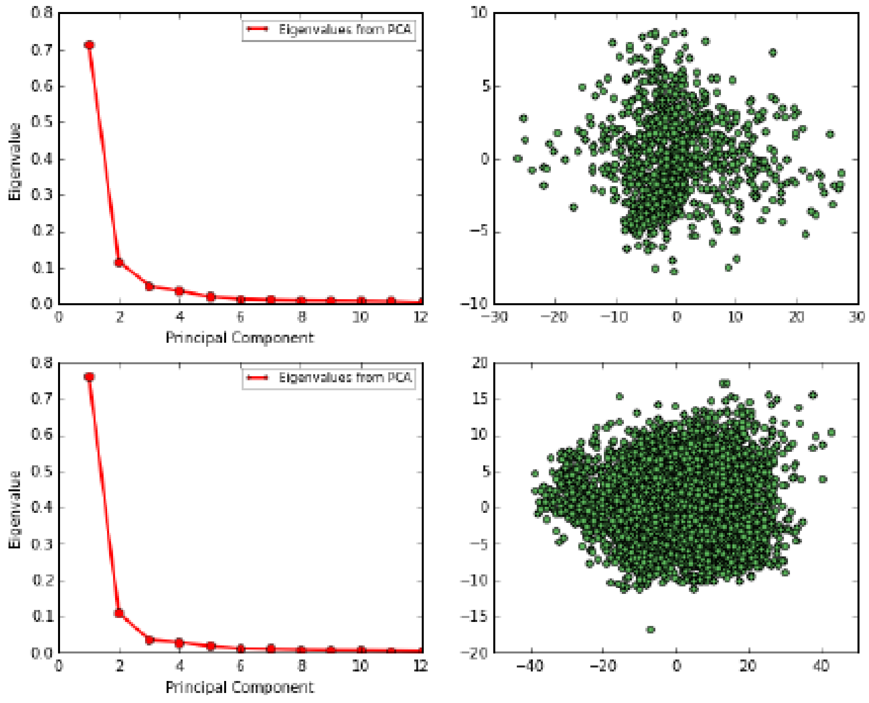
Fig 3.2 PCA - Power



### 3.3.3 Voltage Plot

```
# voltage - v_fields v0, v1, ... v11
vx = raw_df[v_fields].values
vx_pca, v_eigvals = do_pca(vx, 2)
# draw
```

Fig 3.3 PCA - Voltage

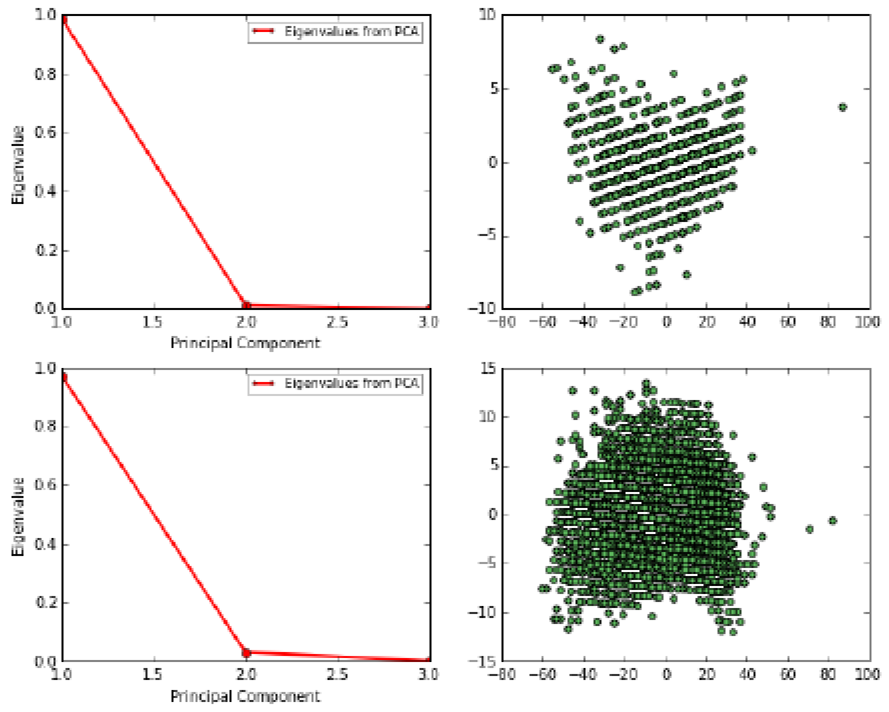


### 3.3.4 Temperature, power, voltage Plot

Then, we feed the the algorithm with the dataset of temperature, power and voltage at a given time point:

```
# 't0', 'k0', 'v0'
x = raw_df[['t0', 'k0', 'v0']].values
x_pca, eigvals = do_pca(x, 2)
# draw
```

Fig 3.4 PCA - Temperature, Power and Voltage



As shown in the plots, the more samples we input, the more regular the data distribution is. We can also find that most points are close to the `(0, 0)` point, and some outlier samples are distributed at the outmost of the group. Clearly the original data mapped to these outlier samples are abnormal values and needed furthe inspection.

## 4. References

[1]. MongoDB - https://en.wikipedia.org/wiki/MongoDB (https://en.wikipedia.org/wiki/MongoDB)
[2]. Open Database Connectivity - https://en.wikipedia.org/wiki/Open_Database_Connectivity (https://en.wikipedia.org/wiki/Open_Database_Connectivity)
[3]. pyodbc 2.1.9 - https://pypi.python.org/pypi/pyodbc/ (https://pypi.python.org/pypi/pyodbc/)
[4]. pymongo 3.0.3 - https://pypi.python.org/pypi/pymongo (https://pypi.python.org/pypi/pymongo)
[5]. Scikit-learn - http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html (http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html)

## 5. Appendix

Online demonstration

```
https://52.27.193.168:8888
password: nbserver
```

main_output.py

```
==================================================================================
The raw csv dataset includes 47 fields sepecified in t_fields, k_fields
and other_fields. Each field in t_fields represents the temperature of
a ammeter at a time point, and the same goes to k_fields and v_fields.
In this case, temperature, power and voltage at twelve time points are
sampled for analysis.

Principal Component Analysis (PCA) applied to this data identifies the
combination of attributes (principal components, or directions in the
feature space) that account for the most variance in the data.
==================================================================================
'''

print(__doc__)

from sklearn.decomposition import PCA, KernelPCA
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

def do_pca(d, n):
    pca = PCA()
    X = pca.fit_transform(d)
    eigvals = pca.explained_variance_ratio_
    pca = PCA(n_components=n)
    X = pca.fit_transform(d)
    print 'eigen value list', eigvals
    print 'chosen ones', pca.explained_variance_ratio_
    return X, eigvals

def draw(dataset, title):
    plt.figure(figsize=(10, 8))
    plt.title(title)
    for index, data in enumerate(dataset):
        eigvals, df = data
        sing_vals = np.arange(len(eigvals)) + 1
        plt.subplot(2, 2, index * 2 + 1)
        plt.plot(sing_vals, eigvals, 'ro-', linewidth=2)
        plt.xlabel('Principal Component')
        plt.ylabel('Eigenvalue')
        leg = plt.legend(['Eigenvalues from PCA'], loc='best', borderpad=0.3,
                         shadow=False, prop=matplotlib.font_manager.FontProperties(size='small'),
                         markerscale=0.4)
        leg.get_frame().set_alpha(0.4)
        leg.draggable(state=True)
        plt.subplot(2, 2, index * 2 + 2)
        plt
        color = '#5cb85c'
        plt.scatter(df['x'], df['y'], c=color)

# temperture t0, t1, ... t1
t_fields = ['t' + str(i) for i in xrange(0, 12)]
# kwh k0, k1, ... k11
k_fields = ['k' + str(i) for i in xrange(0, 12)]
# voltage v0, v1, ... v11
v_fields = ['v' + str(i) for i in xrange(0, 12)]
```

```
# other fields
other_fields = ['meter', 'year', 'month', 'day', 'town', 'amiModel']
all_fields = other_fields + t_fields + k_fields + v_fields

csv1000_filename = 'output_1000.csv'
csv10000_filename = 'output_10000.csv'
csv_filename = 'output.csv'
raw_df = pd.read_csv(csv10000_filename)
raw_df1000 = pd.read_csv(csv1000_filename)
raw_df10000 = pd.read_csv(csv10000_filename)

tx1000 = raw_df1000[t_fields].values
tx1000_pca, tx1000_eigvals = do_pca(tx1000, 2)
tx1000_pca_df = pd.DataFrame({'x': tx1000_pca[:, 0], 'y': tx1000_pca[:, 1]})
tx10000 = raw_df10000[t_fields].values
tx10000_pca, tx10000_eigvals = do_pca(tx10000, 2)
tx10000_pca_df = pd.DataFrame({'x': tx10000_pca[:, 0], 'y': tx10000_pca[:, 1]})
draw([(tx1000_eigvals, tx1000_pca_df), (tx10000_eigvals, tx10000_pca_df)], 'PCA -
Temperature')

kx1000 = raw_df1000[k_fields].values
kx1000_pca, kx1000_eigvals = do_pca(kx1000, 2)
kx1000_pca_df = pd.DataFrame({'x': kx1000_pca[:, 0], 'y': kx1000_pca[:, 1]})
kx10000 = raw_df10000[k_fields].values
kx10000_pca, kx10000_eigvals = do_pca(kx10000, 2)
kx10000_pca_df = pd.DataFrame({'x': kx10000_pca[:, 0], 'y': kx10000_pca[:, 1]})
draw([(kx1000_eigvals, kx1000_pca_df), (kx10000_eigvals, kx10000_pca_df)], 'PCA - Power')

vx1000 = raw_df1000[t_fields].values
vx1000_pca, vx1000_eigvals = do_pca(vx1000, 2)
vx1000_pca_df = pd.DataFrame({'x': vx1000_pca[:, 0], 'y': vx1000_pca[:, 1]})
vx10000 = raw_df10000[t_fields].values
vx10000_pca, vx10000_eigvals = do_pca(vx10000, 2)
vx10000_pca_df = pd.DataFrame({'x': vx10000_pca[:, 0], 'y': vx10000_pca[:, 1]})
draw([(vx1000_eigvals, vx1000_pca_df), (vx10000_eigvals, vx10000_pca_df)], 'PCA - Voltage')

x1000 = raw_df1000[t_fields].values
x1000_pca, x1000_eigvals = do_pca(x1000, 2)
x1000_pca_df = pd.DataFrame({'x': x1000_pca[:, 0], 'y': x1000_pca[:, 1]})
x10000 = raw_df10000[t_fields].values
x10000_pca, x10000_eigvals = do_pca(x10000, 2)
x10000_pca_df = pd.DataFrame({'x': x10000_pca[:, 0], 'y': x10000_pca[:, 1]})
draw([(x1000_eigvals, x1000_pca_df), (x10000_eigvals, x10000_pca_df)], 'PCA - Temperature,
Power and Voltage')
```