Office Appliance Classification From Disaggregated Plug-Load Data

Gerrit De Moor, Elissa Goldner, Brock Petersen
Department of Civil and Environmental Engineering, Atmosphere/Energy Program
Stanford University, Stanford, CA, 94305, USA

***Abstract*—As buildings consume 40% of US primary energy, energy efficiency through smart plug-load management can have a significant impact on US carbon emissions. In this paper we investigate learning algorithms to classify plug-loads based on minutely energy plug-load data and hourly weather data. One month of data is preprocessed into 31 daily features which are used to classify 5660 training samples into 31 labels using a range of multi-class classification algorithms. Of all models tested, random forest trees performed the best, with a test error of 17%. This promising result can likely be further refined with improved feature selection, and a larger dataset. Future work will focus on increasing the model performance and integrating it into a useful product for plug-load management companies.**

I. Introduction

Energy consumption from carbon intensive energy resources is a major contributor to anthropogenic climate change. While transportation and industry contribute to this problem, buildings represent the largest energy consuming end use in the United States at approximately 40% of primary energy.

Historically building energy reduction focused on HVAC and lighting. As those loads have decreased, the importance of appliance loads in buliding energy use has increased. Office buildings represent approximately 50% of the building energy use in the United States and appliances consume 22% of that energy.

Enmetric is a plug-load analysis company that grew out of the need to give facility managers control over the appliance energy use in office buildings. Recognizing the challenge faced by increased energy use, Enmetric hopes to reduce appliance energy use by eliminating energy consumption when appliances are providing no service to the building occupants. Enmetric utilizes smart power strips that sense the appliance load in virtual real-time. Using the smart power strips in conjunction with a software platform, building managers can view the buildings disaggregated appliance energy consumption and create rules to determine when plugs should power down.

Learning more about Enmetrics system, the group identified two challenges that could be assisted by the use of machine learning algorithms; classify the appliance plugged in to a socket to detect when the appliance is moved and predict the type of appliance that is plugged into a socket without having to manually transcribe the appliance type. This paper will explore different techniques to accomplish those goals.

II. Model Function

The main goal of our model is classification of disaggregated loads. Currently, Enmetrics labels each disaggregated load by hand during installation of the smart power strips. If the devices are moved, it is the responsibility of the building manager to report the move to Enmetric. Since Enmetrics data analytics depend on accurate labeling of devices and because it is often difficult for the building manager to keep track of individual employees moving loads around, Enmetric asked the group to investigate an accurate method to detect if a device had been unplugged or moved. The model developed in this project learns device signatures from past data and classifies each load as a particular device on a daily basis. By comparing yesterdays device classifications to todays device classifications, loads that have been classified as a new type can be flagged for review.

III. Data

Enmetric System smart power strips system takes plug-load measurements every second but aggregates the measurements to be stored at the minute level. The energy data collected included average power, minimum and maximum power during the second, average frequency, average current, and average power factor (see table below). For our project, we obtained the minutely data for over 200 appliances in a 50,000 square foot office building in Silicon Valley. The data covered one year from 12/1/2013 to 11/31/2014 (20 GB). Because of computational limits, we were restricted to only analyzing one month at a time (2 GB).

To capture the effects weather has on the appliance energy use, hourly weather data was incorporated in the analysis as well. The Meso West weather database provided hourly temperature and relative humidity for Moffet Airfield.

IV. Features and Preprocessing

All the numerical data provided by Enmetric was processed into features for our models. Since minutely appliance energy profiles vary significantly during the day, all data was aggregated and classified at the day-level for each unique socket. Daily means and standard deviations of the minutely plug-load data were calculated and serve as features.

Temporal features in the form of day of week, day of year, month, and weekend served as additional features. All values (except for weekend, which is binary) were normalized to within the range [-1, 1] by using a cosine and sine to account for the periodic nature of these feature inputs in the algorithms.

Hourly weather data, temperature and humidity, were processed in a similar manner to the Enmetric data. The daily mean, minimum, maximum and standard deviation were calculated for the temperature and humidity. This brings the total feature count to 27 and the number of training samples to 5660 (for one month of data). An overview of the data used and how it was processed into features is shown in the table above.

The plug-load data contained 31 different appliances, resulting in 31 labels to be used for classification. An overview of the labels is shown is in the table below.

DATA	FEATURES	
Minutely Data from Enmetrics	**Daily energy metrics used as features**	
Power Avg.	Mean Power Avg.	Stdev. Power Avg.
Power Min.	Mean Power Min.	Stdev. Power Min.
Power Max.	Mean Power Max.	Stdev. Power Max.
Energy Used	Mean Energy Used	Stdev. Energy Used
Frequency Avg.	Mean Frequency Avg.	Stdev. Frequency Avg.
Voltage Avg.	Mean Voltage Avg.	Stdev. Voltage Avg.
Current Avg.	Mean Current Avg.	Stdev. Current Avg.
Power Factor Avg.	Mean Power Factor Avg.	Stdev. Power Factor Avg.
Temporal Data	**Daily temporal metrics used as features**	
Day of year	Cosine(day of year)	Sine(day of year)
Day of week	Cosine(day of week)	Sine(day of year)
Month	Cosine(month)	Sine(month)
Weekend	Weekend (binary, 0 or 1)	
Hourly weather Data from Meso West	**Daily weather metrics used as features**	
	Mean temperature	Stdev. temperature
Temperature	Max. temperature	Min. temperature
	Mean humidity	Stdev. humidity
Humidity	Max. humidity	Min. humidity

Device Labels		
Battery Charger	Lamp	Projecter
Bridge	Laptop	Refrigerator
Coffe Grinder	Mecho Shade	Scanner
Coffee	Monitor	Speakers
Desktop Computer	Mouse	Stapler
Electronic Picture Frame	Paper Shredder	UPS
External Hard Drive	Pencil Sharpener	USB Hub
Fan	Phoner Charger	Weather Station
Fax	Polycom	Windows Power Upper
Headset	Power Strip	
Kettle	Printer	

V. Classification Models

All learning algorithms considered for use for our project fell under the category of supervised learning methods for classification. Because the dataset included several unique labels, the project is a multiclass classification problem. Several models were tested with varying parameters to find which algorithm resulted in the highest performance rates. The following popular multiclass classification algorithms were tested for our purposes: multiclass support vector machine (SVM standard), multi-class SVM one-vs-one (SVM OvO), multi-class SVM one-vs-rest (SVM OvR), Random Forest, and Gradient Tree Boosting. A more detailed description of each algorithm, as well as its strengths and weaknesses with respect to this project, can be found below. All algorithms were implemented in Python, using the sci-kit learn library.

The tuning of the parameters for each algorithm was done by k-fold cross validation, with a k-value of 3. This rather small value of k was chosen because we had sufficient training samples and we wanted to speed up processing times. The training samples were first randomly permuted before splitting up the data. In this sense, it is very similar to the classic 30% test data, 70% training data rule, but with the difference that we perform this 3 times with random permutations of the dataset.

The trade off between training and testing error for different values of K is shown below for the Random Forest model. The variance versus biased nature of the K value is shown. The graph also supports our selection of a K value of 3.

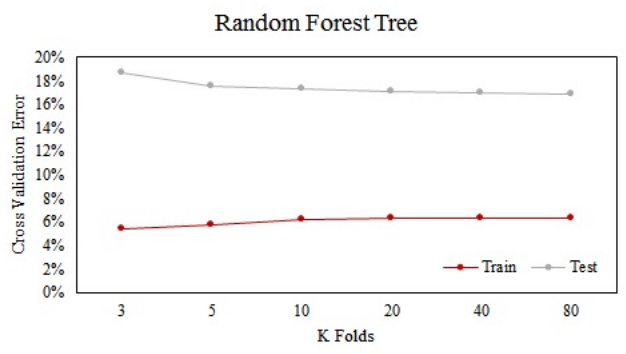

Fig. 1: Training and testing error from random forest for different k values.

VI. Support Vector Machines

Support vector machines (SVMs) are a set of supervised learning algorithms commonly used for classification. SVMs are useful for this particular problem because they are very effective in a high dimensional space. Because they allow the user to choose the kernel function to act as the decision function in the model, SVMs are also flexible to what types of classification problems they are suited for. Despite their effectiveness and versatility, SVMs become less effective when the number of features is much greater than the number of samples. Since this was not the case for our classification problem (31 labels, 5660 training samples), the group started out by classifying using Support Vector Classification (SVC) with a linear kernel and varied different parameters until we found our best results. The general algorithm for a support vector machine is as follows:

Given training vectors $x_i \in R^{p,i=1,\ldots,n}$, in two classes, and a vector $y \in R^n$ such that $y_i \in \{1, -1\}$, SVC solves the following primal problem:

$$\min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1,n} \zeta_i \tag{1}$$

subject to:

$$y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i, \tag{2}$$

$$\zeta_i \geq 0, i = 1, ..., n \tag{3}$$

Its dual is

$$\min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \tag{4}$$

subject to:

$$y^T \alpha = 0 \tag{5}$$

$$0 \leq \alpha_i \leq C, i = 1, ..., l \tag{6}$$

where e is the vector of all ones, C ¿ 0 is the upper bound, Q is an n by n positive semidefinite matrix, $Q_{ij} \equiv K(x_i, x_j)$ and $\phi(x_i)^T \phi(x)$ is the kernel [4].

There are two types of approaches for multi-class SVMs: One-vs-the-Rest (OvR) and One-vs-One (OvO). One-vs-the-Rest considers training a single classifier per class while the One-vs-One trains N(N-1) binary classifiers for a N-way multi-class problem [5]. A more detailed description of each method can be found in the subsequent sections. Research has shown that OvR and OvO are among the most suitable methods for practical use [3].

A. *One-Versus-One SVM*

The first type of SVM that was used for this project was the standard SVC with a linear kernel found in the sci-kit learn package. By default, SVC implements the one-versus-one approach to multi-class classification. OvO classification will build N(N-1) classifiers, one classifier to distinguish between each pair of classes *i* and *j* by marking *i* as the positive example and *j* as the negative. When it is time to make a prediction for a particular unseen sample, all classifiers are applied to the sample and the class with the highest number of positive predictions will be chosen.

$$rbfkernel : \exp(-\gamma |x - x'|^2) \tag{7}$$

For the OvO SVM, we started by tuning the penalty parameter of the error term (C), which can also be seen as the hyperplane separation term. While initially, increasing C lead to smaller prediction errors, values larger than 10 resulted in overfitting the training sample and hence an increase in testing error. It was expected that small values of C would result in more misclassification, but we found that imporvements reached their limit as C approaced a value of 10.The best results for our model were found by using a radial basis function (RBF) kernel, also know as the Gaussian kernel. Varying values of gamma, the kernel coefficient used with RBF kernels, produced no improvement in predictions. A graphical view of the effects of varying C values on the OvO support vector classification model can be found in Figure 2.

B. *One-Versus-The-Rest SVM*

As opposed to OvO where a sample is marked as in class *i* and not in class *j*, One-vs-the-Rest will build N different binary classifiers to determine if the example is either in class *i* or not in class *i* [5]. For SVM OvR, the outputs and ideal tuning parameters were very similar to standard SVM, as seen in Figure 3. Similar to OvO SVM, the ideal parameters for the OvR SVM were found to be C=10 with a RBF kernel.

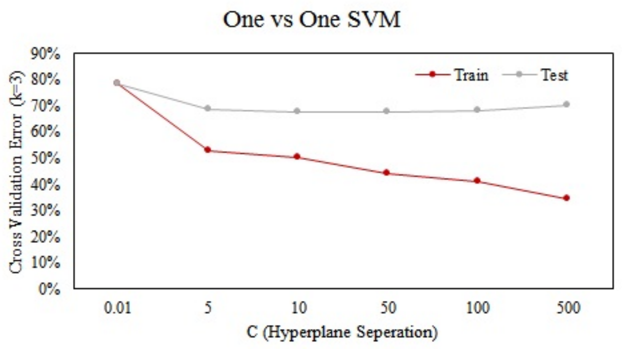

Fig. 2: Training and testing error from One-versus-One support vector classification when varying C value.

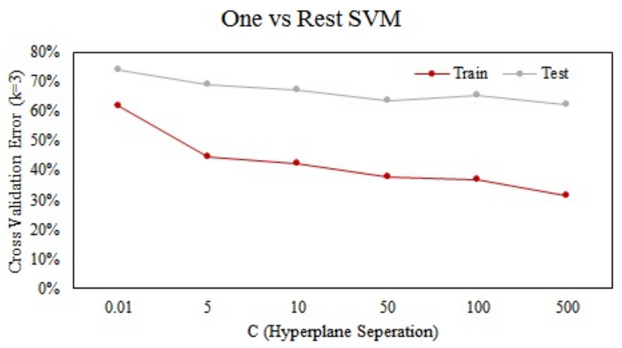

Fig. 3: Training and testing error from One-versus-the-Rest support vector classification when varying C value.

VII. Ensemble Methods

Since SVM results were still showing significant error rates, the group decided to investigate ensemble methods for classification. The goal of ensemble methods is to combine the predictions of several base estimators built with a given learning algorithm to improve the robustness over a single estimator [4]. By combining multiple hypotheses that may produce weak results for a particular classification problem, ensembles methods aim to build a stronger, more accurate hypothesis tailored for the individual problem. The two types of ensemble methods used for our classification problem were random forest classification and gradient tree boosting.

A. *Random Forest*

The random forest classification method falls under the category of an textitaveraging ensemble method. The driving principle behind averaging methods is to build several estimators independently and to average the resulting predictions together in order to reduce the variance in predictions [4]. In random forest classification, a series of randomized decision trees in the ensemble are built from a random subset of the training set. Each subset used is a sample drawn from the feature vector that is replaced after use for continued use in subsequent tests. When splitting a node in the construction of the tree, instead of choosing the best split that is best fit for the entire training set, the split chosen is the best fit among the subset of data. Although the randomness of these subsets usually produces results with higher bias, the nature of the averaging ensemble method also decreases the variance among results, typically much greater than the increase in bias.

When running random forest classification, we varied the the number of estimators and maximum depth used in the algorithm until we found optimal results. Intuitively, we expected the performance to increase as we increased the number of estimators (i.e. the number of trees), but this came with the trade-off of a longer computational time. As can be seen in Figure 4, the improvement in training and testing error as we increased the number of estimators was not as great as we had expected. Surprisingly for the group, varying the max depth of the trees had a much greater effect on performance, as seen in Figure 5. The optimal values of max depth and n-estimators found for our model were 17 and 100, respectively.

Given training vectors $x_i \in R^{n,i=1,\cdots}$, l and a label vector $y \in R^l$, a decision tree recursively partitions the space such that the samples with the same labels are grouped together. Let the data at node m be represented by Q. For each candidate split $\theta = (j, t_m)$ consisting of a feature j and threshold t_m, partition the data into $Q_{left}(\theta)$ and $Q_{right}(\theta)$ subsets

$$Q_{left}(\theta) = (x, y)|x_j <= t_m$$

$$Q_{right}(\theta) = Q \setminus Q_{left}(\theta)$$

The impurity at m is computed using an impurity function H(), the choice of which depends on the task being solved (classification or regression)

$$G(Q,\theta) = \frac{n_{left}}{N_m} H(Q_{left}(\theta)) + \frac{n_{right}}{N_m} H(Q_{right}(\theta))$$

Select the parameters that minimises the impurity

$$\theta^* =_\theta G(Q,\theta)$$

Recurse for subsets $Q_{left}(\theta^*)$ and $Q_{right}(\theta^*)$ until the maximum allowable depth is reached, $N_m < \min_{samples} or N_m = 1$.

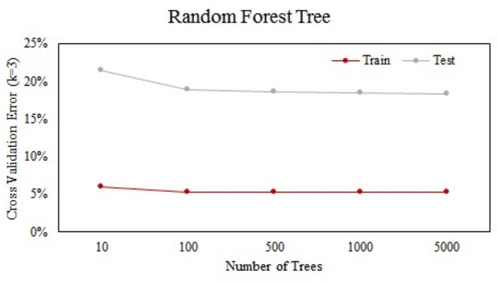

Fig. 4: Training and testing error from random forest classification when varying number of estimators.

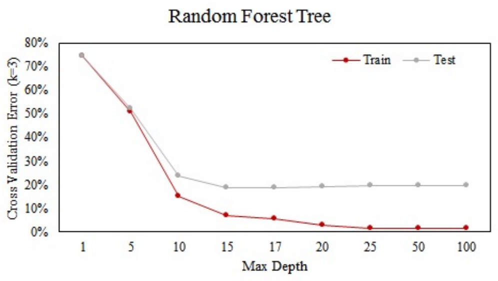

Fig. 5: Training and testing error from random forest classification when varying max depth.

B. Gradient Tree Boosting

The first of two algorithms that we looked at is called gradient boosting classification. Gradient tree boosting combines weak learning methods in an iterative manner until a single, stronger learning method is created. The gradient tree boosting method was chosen for our classification problem because it is typically very good at handling data with heterogeneous features. More concern with this algorithm was around the scalability of its results; although it proved effective for our data sets in month-long intervals, it was thought that combining that data into a full year could cause issues due to the sequential nature of boosting limiting its ability to be parallelized [4].

Since gradient tree boosting is another ensemble method, the group expected that varying the number of estimators and the max depth of the tree branches would act in similar manners as they did in the random forest algorithm. As seen in Figures 6 and 7, our assumptions were largely correct. More interesting however, was the effect that learning rate had on the model results, shown in Figure 8. After varying the parameters in the gradient tree boosting algorithm, the optimal values of n-estimators, learning rate, and max depth found for our model were 60, 0.1, and 10, respectively.

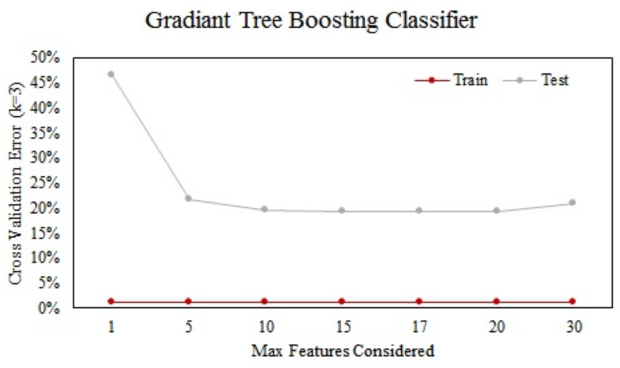

Fig. 6: Training and testing error from gradient tree boosting classification when varying number of estimators.

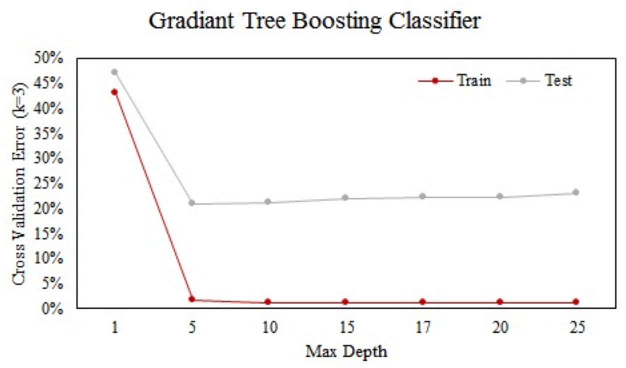

Fig. 7: Training and testing error from gradient tree boosting classification when varying max depth.

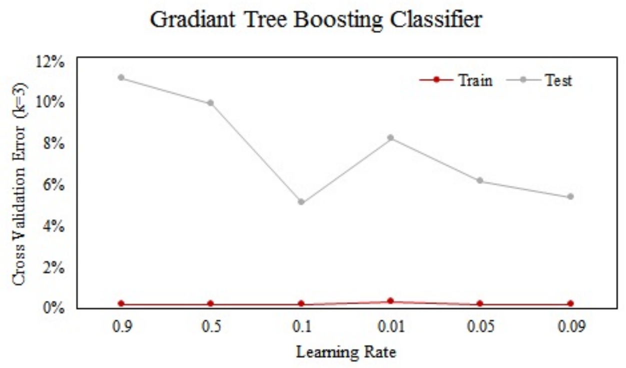

Fig. 8: Training and testing error from gradient tree boosting classification when varying learning rate.

VIII. Results

All three SVM algorithms produced the greatest training and test errors seen out of the five algorithms used. Trying different binary SVM decompositions (OvO and OvR) did not improve the performance. The Random Forest algorithm performed the best in terms of speed and test error. The algorithm ran in under 15 seconds and resulted in a test error of 17%. Gradient Tree Boosting resulted in the lowest training error and performed only slightly worse than the Random Forest algorithm in terms of test error.

The graphs show that tuning the main parameters of the algorithms has a significant impact on the test error. Biased-variance trade off does show in some of the tuning, as can be expected when tuning certain parameters such as the C parameter in SVM models.

The performance of the Random Forest confirmed the fact that random forests usually work faster, but need a sufficient dataset size to work its randomization concept.

IX. Conclusion

SVM performed worst of the algorithms we tested. Trying different binary SVM decompositions (one-vs-one and one-vs-the-rest) did not improve the performance. The Random Forest algorithm performed the best in terms of speed and test error. The algorithm ran in under 15 seconds and resulted in a test error of 17%. Gradient Tree Boosting resulted in the lowest training error and performed only slightly worse than the Random Forest algorithm in terms of test error.

Our work has shown that machine learning algorithms can be a useful, efficient tools in classification of office appliances from plug load data. Future work should be able to further improve the classification to more acceptable levels of test errors. In particular, improved feature extraction and selection, and a larger dataset look promising.

X. Continued Work

The work presented in this paper shows that learning algorithms can be an effective tool for Enmetrics Systems to use in their smart plug analytic services. The group will be presenting our findings at Enmetric Systems next month with the intentions of continuing further and producing a useful tool for the company to possibly use. Based on the successful results developed throughout the course of this project, the following are potential actions that may be taken in the continuation of the project:

1) Run the five learning algorithms presented in this poster on the larger dataset, varying the necessary parameters to find the best fit model for classification.
2) Develop an iterative algorithm that learns based on a collection of past data, predicts for the day, and compares to known to identify any location where deviced could have been moved or unplugged.
3) Explore the space of prediction to eliminate the need for Enmetric to record by hand the type of device at each plug at installation.

Acknowledgment

We would like to thank Jake Masters and Brad Degnan of Enmentric Systems, Inc. for providing us with the data and motivation for this project. We would also like to thank Romain Juban of C3 Energy for his advice and guidance on the use of learning algorithms with load disaggregation and classification. Finally, we would like to thank Professor Andrew Ng and the CS229 teaching and project assistants for the help, guidance, and encouragement.

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.
[2] Machine Learning Applications for Load, Price and Wind Power Prediction in Power Systems, Michael Negnevitsky, Senior Member, IEEE, Paras Mandal, Member, IEEE, and Anurag K. Srivastava, Member, IEEE
[3] H. Lei and V. Govindaraju, *Half-Against-Half Multi-class Support Vector Machines*, State University of New York at Buffalo, http://blue.utb.edu/hlei/Research/papers/Half_HalfSVM.pdf
[4] F. Pedregosa, *et.al*, *Scikit-learn: Machine Learning in Python*, Journal of Machine Learning Research, vol. 12, pg. 2825-2830, 2011
[5] http://www.mit.edu/~9.520/spring09/Classes/multiclass.pdf
[6] http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1583738&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D1583738

APPENDIX A
OFFICE APPLIANCE CLASSIFICATION MODEL

Different versions of this simulation were used throughout this project. This version utilizes the most recent version of the Office Appliance Classification algorithm. This script was written in python, utilizing the pandas and sci-kit learn libraries.

Listing 1: code 1

```
# coding: utf-8

# ###DATA INPUT
# Read in the data and index by time
    stamps
#

# In[1]:

# import warnings
# warnings.filterwarnings("ignore",
    category=DeprecationWarning)


# In[21]:

import pandas as pd
import numpy as np
import sys
import string
from dateutil import parser

input_path = '/Users/Gerrit/Documents/'

#read in all data files from directory
# filenames = [1312, 1401, 1402, 1403,
    1404, 1405, 1406, 1407, 1408, 1410,
    1411]
# df = pd.concat([pd.read_csv(input_path+"
    x_sample_"+str(filename)+".csv",
    low_memory=False) for filename in
    filenames],0)

#read in one data file, only first nrows
# filename = 'x_sample_1401'
filename = 'x_sample_1312'
df = pd.read_csv(input_path+filename+".csv
    ",low_memory=False)


# ### Data manipulation
# Here we play around with the data, and
    set up the X and Y vector

# In[22]:

df.head()

# In[23]:

#create one column of unique IDs from the
    node Hid and channel number
df['channel_number'] = df['channel_number'
    ].apply(str)
df['node_hid'] = df['node_hid'].apply(str)
df['id'] = df.apply(lambda x: x['node_hid'
    ]+ x['channel_number'], 1)
df['id'] = df['id'].apply(int)
df = df.drop(['node_hid', 'channel_number'
    ], 1)

df.head()


# In[24]:

#strip the name column of location numbers
    and call it new_name
df['new_name'] = df.apply(lambda x: x['
    name'],1)
dict_data = pd.read_csv('/Users/Gerrit/
    Documents/Dictionary.csv')
dictionary = dict(zip(dict_data.Old_Name,
    dict_data.New_Name))
df = df.replace({'new_name':dictionary})

#take out unlabeled data
df = df[df.new_name != 'Unknown']
df.head()


# In[25]:

df['occurred_at'] = pd.to_datetime(df['
    occurred_at'])
df['DAY'] = df['occurred_at'].apply(lambda
    x : x.timetuple().tm_yday)
df['COS_DAY'] = np.cos(df['DAY']/365. * 2*
    np.pi)
df['SIN_DAY'] = np.sin(df['DAY']/365. * 2*
    np.pi)
df['WEEKDAY'] = df['occurred_at'].apply(
    lambda x : x.weekday())
df['COS_WEEKDAY'] = np.cos(df['WEEKDAY'
    ]/7. * 2*np.pi)
df['SIN_WEEKDAY'] = np.sin(df['WEEKDAY'
    ]/7. * 2*np.pi)
df['MONTH'] = df['occurred_at'].apply(
    lambda x : x.month)
df['COS_MONTH'] = np.cos(df['MONTH']/12. *
    2*np.pi)
df['SIN_MONTH'] = np.sin(df['MONTH']/12. *
    2*np.pi)
df['WEEKEND'] = (df['WEEKDAY'] > 4)*1
```

```
# In[26]:

calendar_features = ["COS_DAY","SIN_DAY","
    COS_WEEKDAY","SIN_WEEKDAY","COS_MONTH"
    ,"SIN_MONTH","WEEKEND"]
electric_features = ["power_avg", "
    power_min", "power_max", "energy_used"
    , "frequency_avg", "voltage_avg", "
    current_avg", "power_factor_avg"]


# In[27]:

df_mean = df.groupby(['id','DAY'])[
    electric_features].apply(lambda x : x.
    mean())
df_mean.columns = [x+"_mean" for x in df[
    electric_features].columns]
df_std = df.groupby(['id','DAY'])[
    electric_features].apply(lambda x : x.
    std())
df_std.columns = [x+"_std" for x in df[
    electric_features].columns]
df_calendar = df.groupby(['id','DAY'])[
    calendar_features].apply(lambda x : x.
    mean())
df_label = df.groupby(['id','DAY'])[['
    new_name']].apply(lambda x : x.iloc
    [0])


# In[81]:

import numpy as np

df_avg_day = df.groupby(['id','DAY'])[
    electric_features].apply(lambda x : x
    [480:1080].mean())
df_avg_night = df.groupby(['id','DAY'])[
    electric_features].apply(lambda x : x
    [0:480 and 1081:1339].mean())
df_night_day = df_avg_day / df_avg_night
df_night_day.columns = [x+"
    _night_day_ratio" for x in df[
    electric_features].columns]

df_max = df.groupby(['id','DAY'])[
    electric_features].apply(lambda x : x.
    max())
df_max.columns = [x+"_max" for x in df[
    electric_features].columns]

df_min = df.groupby(['id','DAY'])[
    electric_features].apply(lambda x : x.
    max())
df_min.columns = [x+"_min" for x in df[
    electric_features].columns]

# other stuff based on
# Heartbeat Time Series Classification
    With Support
# Vector Machines
# Argyro Kampouraki, George Manis, and
    Christophoros Nikou, Member, IEEE

#    root mean square of successive
    differences
def rmssd(x):
    N = len(x)
     result = np.sqrt(sum(np.ediff1d(x)**2)
        /(N-1))
     return result

df_rmssd = df.groupby(['id','DAY'])[
    electric_features].apply(lambda x :
    rmssd(x))
df_rmssd.columns = [x+"_rmssd" for x in df
    [electric_features].columns]

df_sdsd = df.groupby(['id','DAY'])[
    electric_features].apply(lambda x : np
    .std(np.ediff1d(x)))
df_sdsd.columns = [x+"_sdsd" for x in df[
    electric_features].columns]


# In[82]:

df2 = pd.concat([df_label,df_mean,df_std,
    df_max, df_min, df_night_day, df_rmssd
    , df_sdsd, df_calendar],1)


#### Temperature Data

# In[83]:

weather_data = pd.read_csv(input_path+"
    Weather_Data_Clean.csv")


# In[84]:

weather_data.columns = ['Time','Temp','RH'
    ,'Hour']
weather_data['DAY'] = pd.to_datetime(
    weather_data['Time']).apply(lambda x :
     x.timetuple().tm_yday)
mean_weather = weather_data.groupby(['DAY'
    ])[['Temp','RH']].apply(lambda x : x.
    mean())
```

```
std_weather = weather_data.groupby(['DAY'
    ])[['Temp','RH']].apply(lambda x : x.
    std())
min_weather = weather_data.groupby(['DAY'
    ])[['Temp','RH']].apply(lambda x : x.
    min())
max_weather = weather_data.groupby(['DAY'
    ])[['Temp','RH']].apply(lambda x : x.
    max())

weather = pd.concat([mean_weather,
    std_weather, min_weather, max_weather
    ],1)


# In[85]:

df3 = df2.reset_index().merge(weather, how
    ='left', left_on='DAY', right_index=
    True)


# In[86]:

df3.to_csv(input_path+filename+"_processed
    .csv",index=False)


# In[1]:

# import pandas as pd
# import numpy as np

# input_path = '/Users/Gerrit/Documents/'
# filename = 'x_sample_1312'
# df3 = pd.read_csv(input_path+"
    x_sample_1312_processed.csv")
# df3.set_index(['id','DAY'],inplace=True)


# In[87]:

df4 = np.random.permutation(df3)


# ### Using Machine Learning Algorithms
# Once we have read in the data properly,
    we can use the right machine learning
    algorithm. First we import the sci-kit
     learn (sklearn) library which has all
     the good SVM stuff for this (and all
    other algorithms as well!).

# In[88]:

X = df4[:,1:]
y = df4[:,:1]
```

```
##### Kmeans

# In[20]:

# from sklearn.cluster import KMeans
# import numpy as np



# kmeans = KMeans(n_init = 10)
# kmeans.fit(X)
# labels = kmeans.labels_
# centroids = kmeans.cluster_centers_

# kmeans.cluster_centers_
# max(kmeans.labels_)


##### Support Vector Machine Classifier

# In[8]:

from sklearn.svm import SVC
from sklearn.ensemble import
    RandomForestClassifier
from sklearn.ensemble import
    GradientBoostingClassifier
from sklearn.multiclass import
    OneVsOneClassifier
from sklearn.multiclass import
    OneVsRestClassifier


# In[7]:

k=3
Xcv = np.array_split(X,k)
ycv = np.array_split(y,k)
Xcv_ = [np.vstack(tuple([Xcv[j] for j in
    range(k) if j!=i])) for i in range(k)
    ];
ycv_ = [np.vstack(tuple([ycv[j] for j in
    range(k) if j!=i])) for i in range(k)
    ];


# In[25]:

#Select one model

# C = [3, 5, 10, 20, 40, 80]
C = [10]
# C = [0.1, 5.0, 10.0, 50.0, 100.0, 500.0]

for c in C:
#       model = SVC(C=50, kernel='rbf',
    gamma=0.0)
```

```
#       tuning: C = 50
    model = RandomForestClassifier(
        n_estimators=100, max_depth=17,
        max_features='auto',  n_jobs=4)
#       tuning: n_estimators = 100,
    max_depth = 17
#       model = GradientBoostingClassifier(
    learning_rate=0.1, n_estimators=60,
    max_depth=10, max_features=15)
#       tuning: learning_rate = 0.1,
    estimators = 60, max_depth = 5 (or 10)
    , max_features = 15

    ## To add to SVC model
#       model = OneVsOneClassifier(model,
    n_jobs=4)
#       model = OneVsRestClassifier(model,
    n_jobs=4)
#       tuning: C = 500

    error_train = [[]]*c
    error_test = [[]]*c

#       Xcv = np.array_split(X,c)
#       ycv = np.array_split(y,c)
#       Xcv_ = [np.vstack(tuple([Xcv[j] for
    j in range(c) if j!=i])) for i in
    range(c)];
#       ycv_ = [np.vstack(tuple([ycv[j] for
    j in range(c) if j!=i])) for i in
    range(c)];

    for i in range(c):

        model.fit(Xcv_[i],ycv_[i][:,0])
        #preds_tr = model.predict(Xcv_[i])
        #print "Training error for fold
            {}: {}".format(i+1,1-model.
            score(Xcv_[i],ycv_[i]))
        error_train[i] = 1-model.score(
            Xcv_[i],ycv_[i])
        #preds_te = model.predict(Xcv[i])
        #print "Test error for fold {}:
            {}".format(i+1,1-model.score(
            Xcv[i],ycv[i]))
        error_test[i] = 1-model.score(Xcv[
            i],ycv[i])
    print c, "Training error: {}, Cross-
        validated error: {}".format(np.
        mean(error_train), np.mean(
        error_test))


# In[46]:

prediction = model.predict(X)
print(prediction)
print('Number of mispredictions is %d\n' %
    sum(prediction != y[:,0]))

# np.hstack(tuple(prediction,y[:,0])
np.hstack(tuple([prediction,y[:,0]]))

prediction != y[:0]


# In[60]:

# import matplotlib.pyplot as plt
# %matplotlib inline


# In[65]:

# plt.plot(errors)
# plt.xlabel("C")
# plt.ylabel("Cross-validated error")
# plt.title("Tuning of parameter C")
# #plt.savefig("test")


# In[ ]:
```