Power Regression of Air-conditioner

Mucun Tian

Abstract

In this project, we analyze the power consumption of air-conditioner and its relationship with environmental factors including wind, solar, humidity and temperature. To predict the power consumption, we apply four different kinds of regression models which are linear regression, decision tree, random forest and gradient boosting trees. Each model is trained and tuned through a 5-fold cross validation paradigm, and the absolute relative prediction error is reported on test data. Results show that our models can track the change trends of power consumption.

Background

Energy consumption in buildings takes up to 40% of total energy consumption of U.S. in 2016 [1]. How to model and predict the electricity consumption of buildings plays an important role in energy conservation. The most common methods to handle this problem can be classified into three categories: physical laws, machine learning and statistical methods, and hybrid method [2]. In recent years, machine learning which learns the model of a black box using historical input and output is extensively used in many fields due to high accuracy and simplicity. Popular methods used in energy consumption are multivariate linear regression model (MLR), decision tree and artificial neural networks (ANN) [3, 4].

MLR, due to its computational simplicity and flexibility of configuration, is widely applied in energy consumption, retrofit saving estimation and electricity load forecasting [5-9]. There is another family, tree-based methods including single tree method: decision tree and ensemble trees methods: random forest and gradient boosted trees, which recently shows very good performance of prediction in many domains but seems being less researched in electricity consumption [10]. In this project, we collected residential building's external weather data and used MLR with the elastic net regularization as a baseline, and compared it with decision tree, random forest and gradient boosted trees. To examine the importance and impact of different weather condition, including temperature, humidity, wind and solar, we also did an exhaust experiments with all the combination of weather condition variable to train our models. In the following sections, we will show how we pre-process the data and explore feature importance. We will also introduce our training and hyper-parameter tuning in this project.

Linear Regression

A multivariate linear regression model treats the relationship between response variable (e.g. power) and the explanatory variables (e.g. temperature, humidity, etc.) as a linear combination:

$$y_{\_hat} = wx = W_0 + W_1X_1 + W_2X_2 + \ldots + WnXn,$$

And a loss function is defined as the square of prediction error, $L = || y - wx ||^2$. Then using optimization methods, the optimal value of W which minimizes the loss function can be solved. Since this linearity, it's simple and fast to train. However, a downside is overfitting which means overly learning from the training data and then performing badly on test data. So adding regularization terms which penalizes model complexity can mitigate this weakness. Specifically,

elastic net [13] adds two terms in the loss function:

$$L = ||y - wx||^2 + \text{alpha} * (\text{lambda} * ||W||_1) + (1 - \text{alpha}) * (\text{lambda} / 2 * || W ||_2^2),$$

where alpha belongs to [0,1], and lambda >= 0.

The second term is called L1 term (absolute norm of w) which forces the sparsity of the model by making many terms of w after training be zero and works as a feature selection. And the third term is called L2 term (the square of Euclidean norm) which penalizes complex model (the more number of elements of w the bigger value of this term) and prefers using as less as possible weight terms to model the linearity.

Decision tree

Decision tree [4] is a tree structure where each node represents a feature variable (like temperature, humidity, etc.) which works as a split criterion to divide data into different branches depending on data's value of this feature. Since we are modeling a continuous data, decision tree becomes a regression tree. For a regression tree, major steps are as following:

1. treat each training data point as a vector and the dimension including feature variables(temperature, humidity, etc.) and a label variable(like power), and calculate the variance of the label variable over all data.
2. for each feature variable, compute the quantile based on the value of this feature over all data. And the number of the quantile we computed is the maximal number of bins the data will stay in. Then compute the variance of data within different bins, and get a mean variance based on frequency of the data in each bin.
3. for each feature variable, compute the difference between the feature's mean variance and the label variance. The one with maximal variance reduction is chosen for the first node.
4. program run recursively until stopping criteria is reached. (like maximal depth of the tree or at least samples in each branch)
5. in each branch, the mean of the label variable of training data is used for prediction.

Random forest

Random forest [15] trains many different decision trees separately using sampling with replacement of the training data. At each node, it also only considers random subsets of all feature variables. The mean of predictions from all the trees is used for final prediction. By using more trees than decision trees, random forest reduces the chance of over-fitting and variance of prediction error.

Gradient boosted Trees

Gradient boosted trees [16] is another ensemble trees learning which assigns a weight that will be learned to every leaf node of each tree (each training sample has different output weight of different trees), and instead of using the mean of the label variable as prediction it uses the summation of all the corresponding weight of all trees to predict each sample. The prediction and the label variable can construct a convex loss function (such as square of error). Using the optimization method, the optimal weight can be learned. Gradient boosted trees uses a way similar to stochastic gradient to train the model. It starts from a single tree and learns the optimal weight, then adds a new tree and learns the weight which minimize the loss function, and repeat this process until maximal number of trees reached. For constructing each tree, it also uses same loss function to decide which feature and quantile should be applied to each node to split the training samples.

Data Process

The dataset was gathered over about a month period in the last summer. Each data point is measured by different sensors (solar, wind, humidity, temperature and power) at every 5 seconds. For each day, there are about 17000 data points. We collected data and saved them to five .csv files, which are: wind.csv, Temperature.csv, Solar.csv, humidity.csv, power.csv. For each column, it represents one day's data that are measured every 5 seconds.

First, we combine these data source files:

In [3]:
```
file_names = os.listdir('data')
feature_names = [re.sub(".csv", "", name).lower() for name in file_names]

data_list = []
for i, file_name in enumerate(file_names):
    # read each file
    file_path = os.path.join('data', file_name)
    data = pd.read_csv(file_path)
    # add a sample number column
    data['sample_num'] = range(len(data))
    # convert wide data format to tall format
    data = data.melt(id_vars=['sample_num'], var_name='day', value_name=feature_
names[i])
    data_list.append(data)
# horizontally stack together each variable
data = reduce(lambda x, y: x.merge(y), data_list)
data['day'] = data['day'].map(lambda x : int(re.sub("Day", "", x)))
data.head()
```

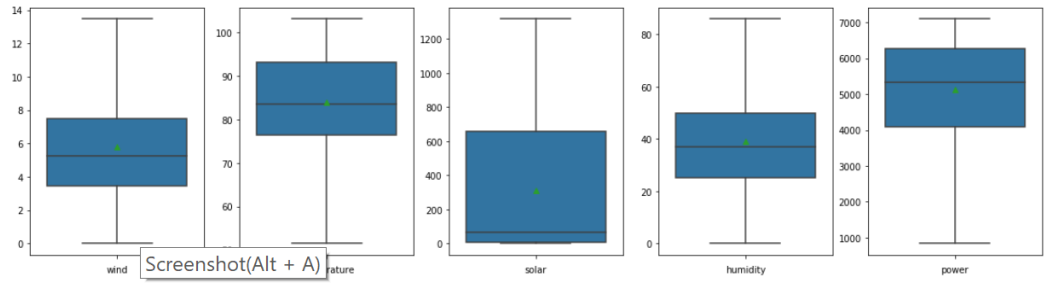
Out[3]:

	sample_num	day	wind	temperature	solar	humidity	power
0	0	1	1.739410	75.128491	1.406047	43.811749	5788.555960
1	1	1	2.189845	73.661096	0.468682	43.245738	5790.605704
2	2	1	2.194219	74.757276	5.757763	43.908135	5771.830051
3	3	1	2.409350	75.669341	8.530939	44.386841	5748.146726
4	4	1	2.230627	75.590562	6.914409	44.364088	5793.202046

Then, we look at the statistics of each variable by Boxplot.

Boxplot

```
In [5]: fig, ax_arr = plt.subplots(1, 5, figsize=((20, 5)))
        variables = ['wind', 'temperature', 'solar', 'humidity', 'power']
        for i, ax in enumerate(ax_arr):
            var = variables[i]
            sns.boxplot(y=var, data=data,
                        showmeans=True, showfliers=False, ax=ax)
            ax.set_ylabel('')
            ax.set_xlabel(var)
        plt.show()
```



We also try to use pandas.DataFrame.describe to generate some descriptive statistics of the data.

```
In [6]: data[variables].describe().transpose()
```

Out[6]:

	count	mean	std	min	25%	50%	75%	max
wind	380107.0	5.797249	3.354542	-8.781063e-15	3.465816	5.255490	7.470503	23.104488
temperature	380107.0	83.924002	12.534455	-4.000000e+01	76.358055	83.630352	93.096285	103.105553
solar	380107.0	313.625594	377.298571	-2.695444e-10	8.110301	67.161087	657.714842	1317.918969
humidity	380107.0	39.217812	16.296445	-3.291698e-14	25.330173	37.159912	50.056655	85.928384
power	380107.0	5138.727569	1248.125151	-4.382715e+01	4090.644955	5328.696748	6264.580691	7099.686172

We can see that there are some negative values in each variable. We will filter out these outliers if they are 1.5 times interquartile range below 25 percentile and above 75 percentile.

First, we define the outlier_mask:

```
In [7]: def outlier_mask(var):
            # compute 25 and 75 percentile
            q25 = var.quantile(.25)
            q75 = var.quantile(.75)
            # interquartile range
            interquartile_range = q75 - q25
            upper = q75 + interquartile_range
            lower = max(q25 - interquartile_range, 0)
            return (var > upper) | (var < lower)
```

Then we filter the outliers out:

Filter out outliers

In [8]:
```
outlier_mask_df = data[variables].apply(outlier_mask)
# As long as there is a outlier in at least one variable, a whole row is thrown away
invalid_data_mask = outlier_mask_df.apply(any, axis=1)
# Filter raw data
data_filtered = data[~invalid_data_mask]
# reset the sample number:
# drop original sample_num colunm,
# groupby day and reset index to use re-index as sample number
# insert the re-index to columns
data_filtered = (data_filtered.drop('sample_num', axis=1)
                 .groupby('day', group_keys=False).apply(lambda g:g.reset_index(drop=True))
                 .reset_index().rename(columns={'index':'sample_num'}))
data_filtered[variables].describe()
```

Out[8]:

	wind	temperature	solar	humidity	power
count	343060.000000	343060.000000	343060.000000	343060.000000	343060.000000
mean	5.272808	85.011782	341.823063	38.267210	5205.153925
std	2.481888	9.269158	384.489884	15.008193	1198.651405
min	0.187760	59.860548	0.000000	11.612810	1916.951821
25%	3.417636	76.841145	8.263144	25.004436	4103.826705
50%	5.088318	84.272376	132.325573	36.550124	5463.625158
75%	7.001496	93.633464	706.414885	49.436258	6304.901203
max	11.475087	103.105553	1306.345520	74.782992	7099.686172

And the new data looks like:

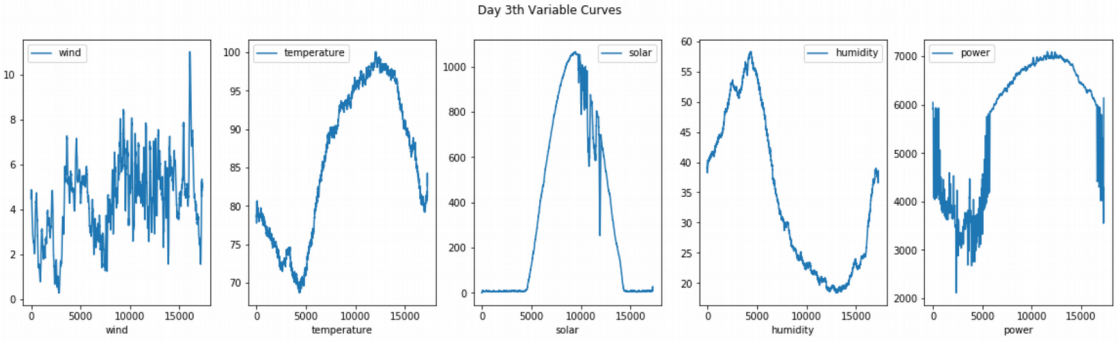
In [9]:
```
data_filtered.head()
```

Out[9]:

	sample_num	day	wind	temperature	solar	humidity	power
0	0	1	1.739410	75.128491	1.406047	43.811749	5788.555960
1	1	1	2.189845	73.661096	0.468682	43.245738	5790.605704
2	2	1	2.194219	74.757276	5.757763	43.908135	5771.830051
3	3	1	2.409350	75.669341	8.530939	44.386841	5748.146726
4	4	1	2.230627	75.590562	6.914409	44.364088	5793.202046

Now we plot one day's data for all variables:

In [10]:
```
day_num = np.random.randint(len(set(data_filtered['day'].values))) + 1
data_day_one = data_filtered[data_filtered.day==day_num]

fig, ax_arr = plt.subplots(1, 5, figsize=((20, 5)))
variables = ['wind', 'temperature', 'solar', 'humidity', 'power']
for i, ax in enumerate(ax_arr):
    var = variables[i]
    data_day_one.plot(x='sample_num', y=var, ax=ax)
#     ax.set_ylabel('')
    ax.set_xlabel(var)
fig.suptitle("Day {}th Variable Curves".format(day_num))
plt.show()
```

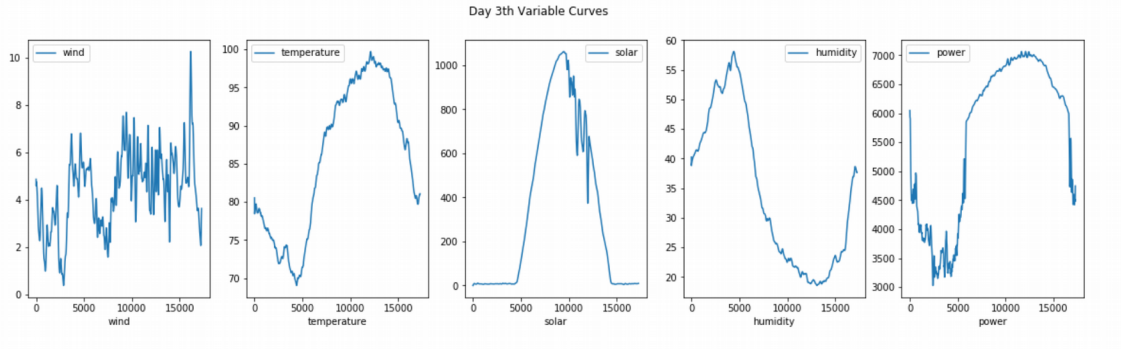


From this plot, we can see that there are some fluctuations for each curve. This might be due to our small sampling period (5 seconds) and the sensor noise.

We can use a mean window to smooth our data curves. Since all the variables we measured shouldn't change very frequently in the outdoor environment, we can use a window size 120 (10 minutes) to smooth our data.

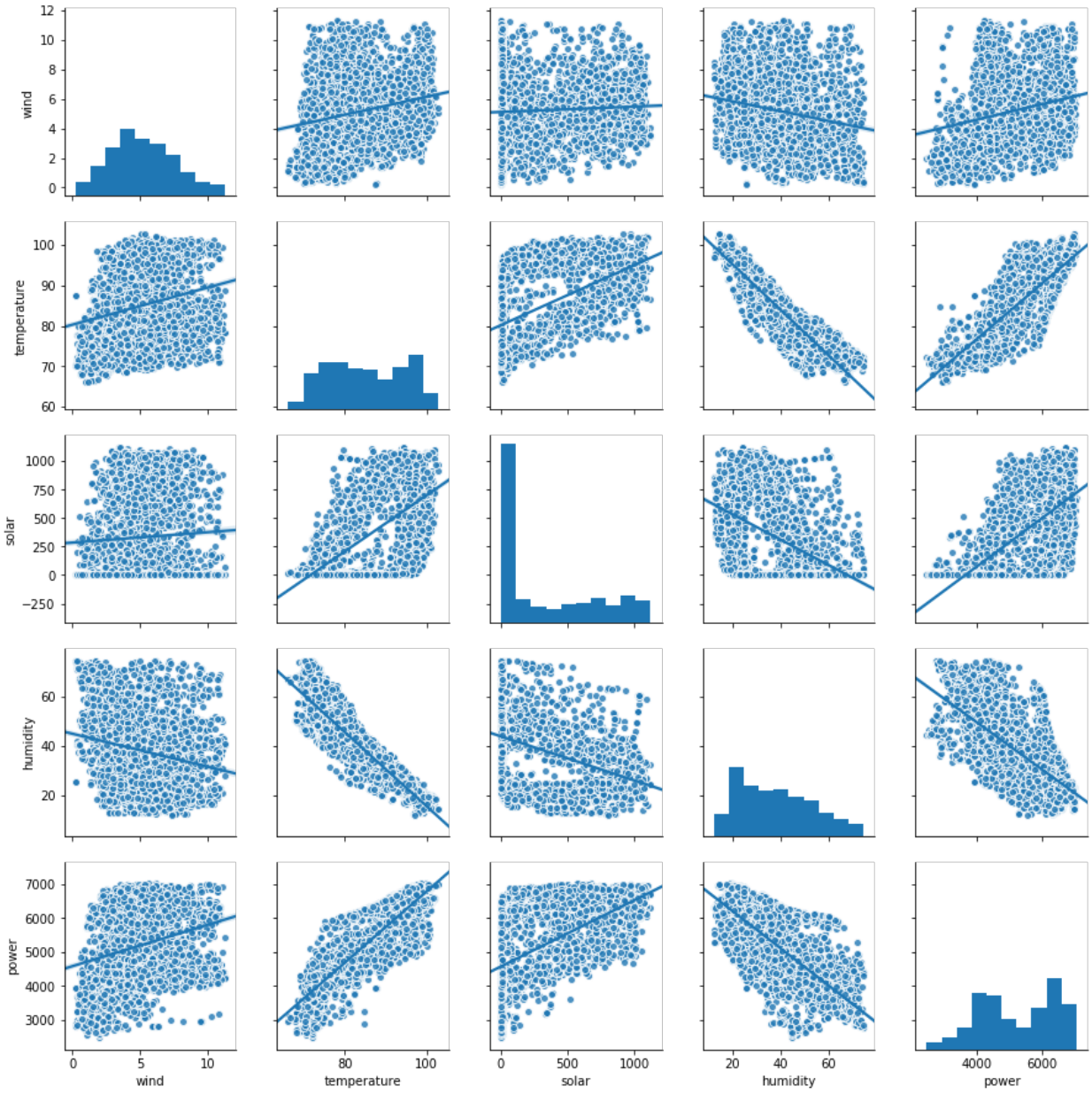
```
In [11]: data_smoothed = (data_filtered.drop('sample_num', axis=1)
                          .groupby('day').rolling(120, 1).mean()
                          .drop('day', axis=1).reset_index(0)
                          .groupby('day', group_keys=False).apply(lambda g:g.reset_index(drop=True))
                          .reset_index().rename(columns={'index':'sample_num'}))
```

```
In [12]: data_day_one = data_smoothed[data_smoothed.day==day_num]
         fig, ax_arr = plt.subplots(1, 5, figsize=((20, 5)))
         variables = ['wind', 'temperature', 'solar', 'humidity', 'power']
         for i, ax in enumerate(ax_arr):
             var = variables[i]
             data_day_one.plot(x='sample_num', y=var, ax=ax)
         #     ax.set_ylabel('')
             ax.set_xlabel(var)
         fig.suptitle("Day {}th Variable Curves".format(day_num))
         plt.show()
```



Feature Exploration

Before building our prediction models, let's look at the correlations between these variables. We sample 10% data and plot the scatter plot for each pair of variables. Since our data points don’t have large variation, 10 percent of the data can be seen as statistically valid for data visualization.



From the pair-plot, we can see that there might be four linear relationship. Two are positive: Temperature vs Power, Solar vs Power and two are negative: Humidity vs Temperature and Humidity vs Power.

To quantify the correlation relationship, we are going to compute the Pearson Correlation Coefficient (PCC).

Feature Importance on Linear Regression

In previous section, we saw that the temperature has strong PCC with power. In this section, we are going to try all the feature combinations and use each combination to train a linear regression model to see how each feature set impacts on the Mean Square Error of predictions. We split data into train and test, and hold out the data of the last two days as the test data. Then the training set is used to tune hyper-parameters through 5-folds cross validation.

```
In [16]: data_train = data[data['day'] <= 20]
         data_test = data[data['day'] > 20]
         X_y_train = data_train[variables].values
         X_y_test = data_test[variables].values
```

Since the scale of these variables are different with each other, this may cause some features values more important than others in the training process. So we need to normalize each variables.

In [17]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler

# Scale features with mean zero and unit standard deviation
# sd = MinMaxScaler()
# sd = StandardScaler()
# sd.fit(X_y_train)
# X_y_train_scaled = sd.transform(X_y_train)
# X_y_test_scaled = sd.transform(X_y_test)
# X_train = X_y_train_scaled[:, :-1]
# y_train = X_y_train_scaled[:, -1]
# X_test = X_y_test_scaled[:, :-1]
# y_test = X_y_test_scaled[:, -1]

X_train = X_y_train[:, :-1]
y_train = X_y_train[:, -1]
X_test = X_y_test[:, :-1]
y_test = X_y_test[:, -1]
```

In [18]:
```
variables
```

Out[18]:
```
['wind', 'temperature', 'solar', 'humidity', 'power']
```

Get all feature combinations

In [19]:
```
import itertools

feature_combinations = []
features = variables[:-1]
for n in range(len(features)):
    feature_combinations.extend(list(itertools.combinations(features, n + 1)))
```

In [20]:
```
feature_combinations
```

Out[20]:
```
[('wind',),
 ('temperature',),
 ('solar',),
 ('humidity',),
 ('wind', 'temperature'),
 ('wind', 'solar'),
 ('wind', 'humidity'),
 ('temperature', 'solar'),
 ('temperature', 'humidity'),
 ('solar', 'humidity'),
 ('wind', 'temperature', 'solar'),
 ('wind', 'temperature', 'humidity'),
 ('wind', 'solar', 'humidity'),
 ('temperature', 'solar', 'humidity'),
 ('wind', 'temperature', 'solar', 'humidity')]
```

For each feature set, we will do a 5-fold cross validation to tune the hyper-parameters, then the best average result across all folds will be used as the metric score of this set of features.

In [21]:
```
from scipy.stats import uniform
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import RandomizedSearchCV

X_train_df = pd.DataFrame(X_train, columns=features)
# parameter grid
alpha = uniform(0, 100)
l1_ratio = uniform(0, 1)
parameters = {'alpha':alpha, 'l1_ratio':l1_ratio}
best_scores = []
for feature_set in feature_combinations:
    feature_set = list(feature_set)
    X_train_features = X_train_df[feature_set].values
    lr_model = ElasticNet(fit_intercept =True)
    clf = RandomizedSearchCV(lr_model, parameters, scoring='neg_mean_squared_error',
                             cv=5)
    clf.fit(X_train_features, y_train)
    best_score = clf.best_score_
    best_scores.append(best_score)
```

Sort the best scores and print the rank

```
In [22]: print(np.array(feature_combinations)[np.argsort([-score for score in best_scores])])
```

```
[('wind', 'temperature', 'solar', 'humidity')
 ('temperature', 'solar', 'humidity') ('wind', 'temperature', 'solar')
 ('temperature', 'solar') ('wind', 'temperature', 'humidity')
 ('temperature', 'humidity') ('wind', 'temperature') ('temperature',)
 ('wind', 'solar', 'humidity') ('solar', 'humidity') ('wind', 'solar')
 ('solar',) ('wind', 'humidity') ('humidity',) ('wind',)]
```

From this rank, we can see the importance rank for features based on prediction errors are:

- Temperature > Solar > Humidity > Wind

Decision Tree Importance

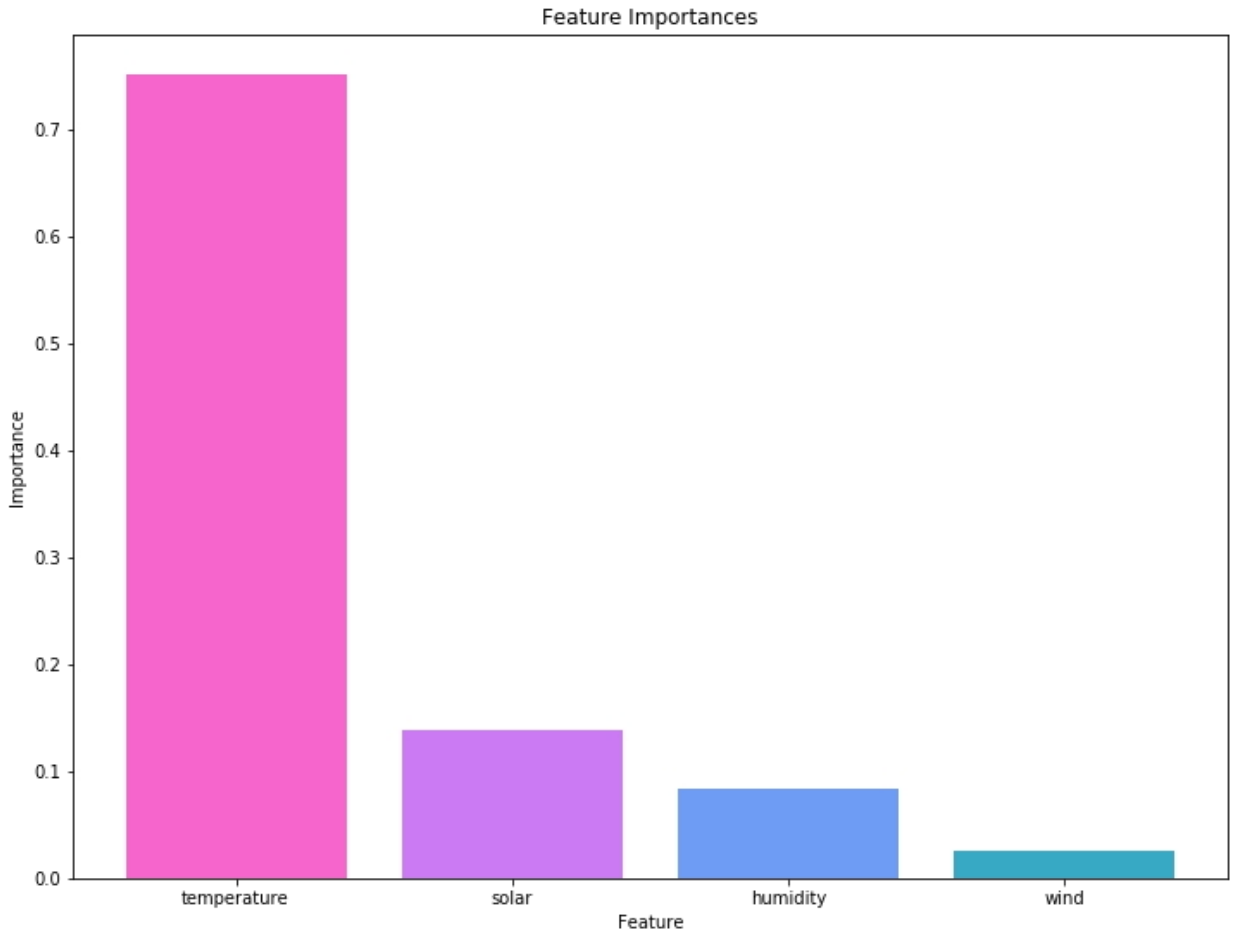
We can also build a decision tree regressor and decide the feature importance based on its Gini importance.

```
In [23]: from scipy.stats import uniform
         from sklearn.tree import DecisionTreeRegressor
         from sklearn.model_selection import GridSearchCV

         # parameter grid
         max_depth = [2, 4, 6, 8]
         parameters = {'max_depth':max_depth}

         dt_model = DecisionTreeRegressor()
         clf = GridSearchCV(dt_model, parameters, scoring='neg_mean_squared_error', cv=5)
         clf.fit(X_train, y_train)

         dt_importance = clf.best_estimator_.feature_importances_
```

```
In [24]: print(np.array(features)[np.argsort([-importance for importance in dt_importance])])
```

```
['temperature' 'solar' 'humidity' 'wind']
```

We see the same rank here. Let’s plot it:



Build Prediction Models

Let's build several regression models. For all the models, we apply 5-fold cross validation and randomized search to tune the hyper-parameters. Then the best model is used to predict the test data and the results are reported. Since we see that several features are linearly correlated with the power variable, we choose the linear regression as our baseline model.

Linear Regression

We tune the L1 ratio in a uniform distribution (0, 1) and the L2 coefficient alpha in an uniform distribution (0, 100)

In [24]:
```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform

alpha = uniform(0, 100)
l1_ratio = uniform(0, 1)

lr_model = ElasticNet(fit_intercept =True)
parameters = {'alpha':alpha, 'l1_ratio':l1_ratio}
clf = RandomizedSearchCV(lr_model, parameters, n_iter=50,
                         scoring='neg_mean_squared_error', cv=5,
                         n_jobs=-1)

clf.fit(X_train, y_train)
lr_best_params = clf.best_params_
lr_best_model = clf.best_estimator_
coefficients = lr_best_model.coef_
intercept = lr_best_model.intercept_
```

In [25]:
```
print(lr_best_params)
```

```
{'alpha': 22.40112759075641, 'l1_ratio': 0.33593277618913897}
```

In [26]:
```
linear_function = str(intercept)
for i, pair in enumerate(zip(features, coefficients)):
    coef = str(abs(pair[1]))
    if pair[1] > 0:
        sym = ' + '
    else:
        sym = ' - '
    var = pair[0]
    linear_function += sym + coef + "*" + var
```

In [27]:
```
print('best parameters:', best_params)
print('Linear Function:', linear_function)
```

The best parameters: {'alpha': 22.40112759075641, 'l1_ratio': 0.33593277618913897}
Linear Function:
power = 1631.24 + 22.1*wind + 44.04*temperature + 1.01*solar – 15.37*humidity

The mean square error on test set is 664.018.

In [28]:
```
from sklearn.metrics import mean_squared_error

y_true = y_test
y_pred = lr_best_model.predict(X_test)
print(np.sqrt(mean_squared_error(y_true, y_pred)))
```

```
664.018272338574
```

Decision Tree

We tune the max depth (2, 4, 8, 10, 12) and use grid search.

In [38]:
```
# parameter grid
max_depth = [2, 4, 8, 10, 12]
parameters = {'max_depth':max_depth}

dt_model = DecisionTreeRegressor()
clf = GridSearchCV(dt_model, parameters, scoring='neg_mean_squared_error', cv=5,
                   n_jobs=-1)
clf.fit(X_train, y_train)
dt_best_params = clf.best_params_
dt_best_model = clf.best_estimator_
```

In [39]:
```
print(dt_best_params)
```

```
{'max_depth': 4}
```

In [40]:
```
y_true = y_test
y_pred = dt_best_model.predict(X_test)
print(np.sqrt(mean_squared_error(y_true, y_pred)))
```

```
717.1928709544985
```

The best parameters are: {'max_depth': 4}
The mean square error on test set is 717.1928709544804.

Random Forest

We tune the max depth (2, 4, 8, 10, 12) and number of trees (10, 20, 50, 80, 100, 150, 200) and use a grid search.

In [34]:
```
from sklearn.ensemble import RandomForestRegressor

# parameter grid
max_depth = [2, 4, 6, 8, 10]
n_estimators = [10, 20, 50, 80, 100, 150, 200]
parameters = {'max_depth':max_depth, 'n_estimators':n_estimators}

rf_model = RandomForestRegressor()
clf = GridSearchCV(rf_model, parameters, scoring='neg_mean_squared_error', cv=5,
                   n_jobs=4)
clf.fit(X_train, y_train)
rf_best_params = clf.best_params_
rf_best_model = clf.best_estimator_
```

In [104]:
```
print(rf_best_params)
```

```
{'max_depth': 4, 'n_estimators': 100}
```

In [35]:
```
y_true = y_test
y_pred = dt_best_model.predict(X_test)
print(np.sqrt(mean_squared_error(y_true, y_pred)))
```

```
717.1928709544804
```

The best parameters are: {'max_depth': 4, 'n_estimators': 100}

The mean square error on test set is 717.1928709544804.

Gradient Boosting Trees

We tune the max depth (2, 4, 8, 10, 12) and number of trees (10, 20, 50, 80, 100, 150, 200) and use a randomized search.

In [96]:
```
from sklearn.ensemble import GradientBoostingRegressor

# parameter grid
max_depth = [2, 4, 6, 8, 10, 12]
n_estimators = [10, 20, 50, 80, 100, 150, 200]
parameters = {'max_depth':max_depth, 'n_estimators':n_estimators}

gb_model = GradientBoostingRegressor()
clf = RandomizedSearchCV(gb_model, parameters, scoring='neg_mean_squared_error', cv=5, n_jobs=4)
clf.fit(X_train, y_train)
gb_best_params = clf.best_params_
gb_best_model = clf.best_estimator_
```

In [105]:
```
print(gb_best_params)
```

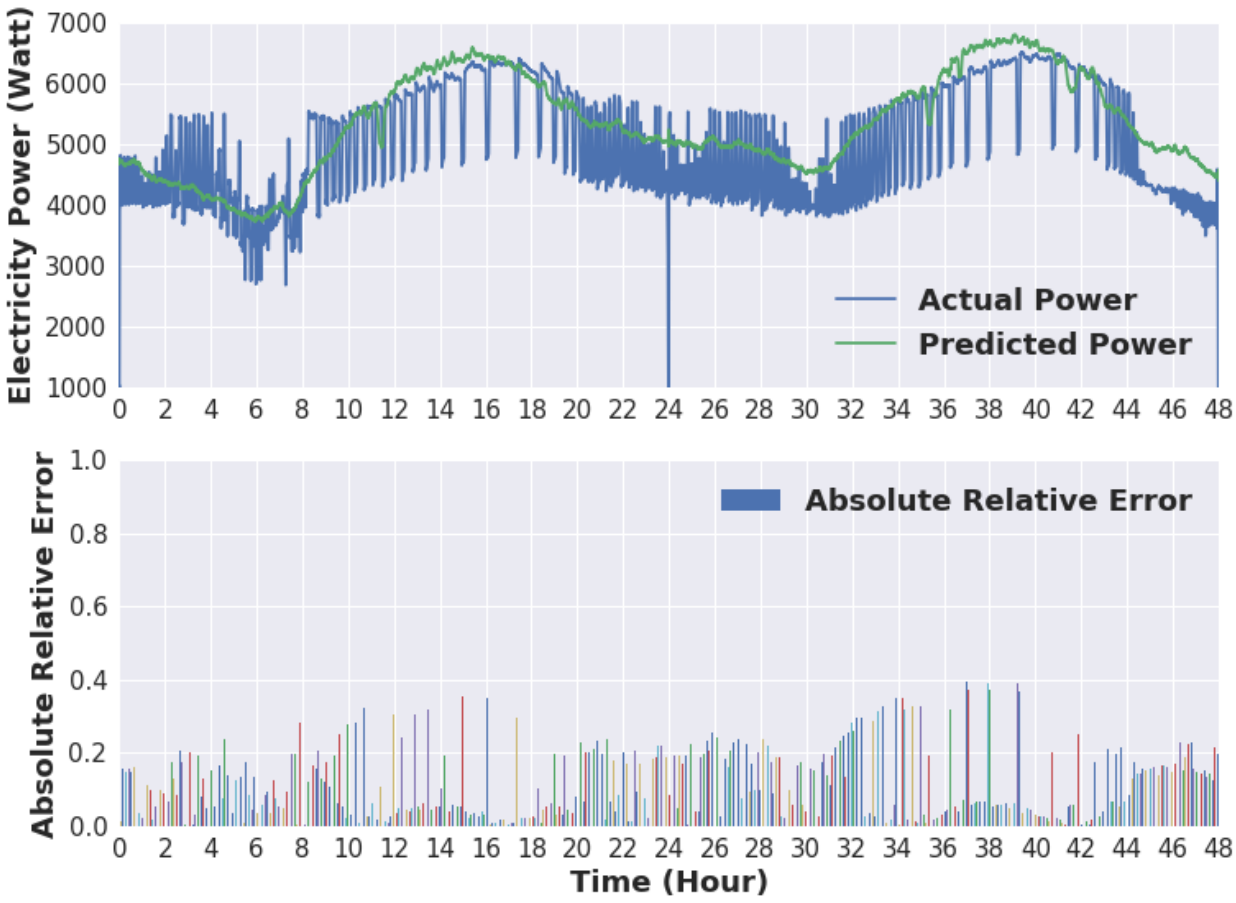
```
{'n_estimators': 80, 'max_depth': 2}
```

In [97]:
```
y_true = y_test
y_pred = gb_best_model.predict(X_test)
print(np.sqrt(mean_squared_error(y_true, y_pred)))
```

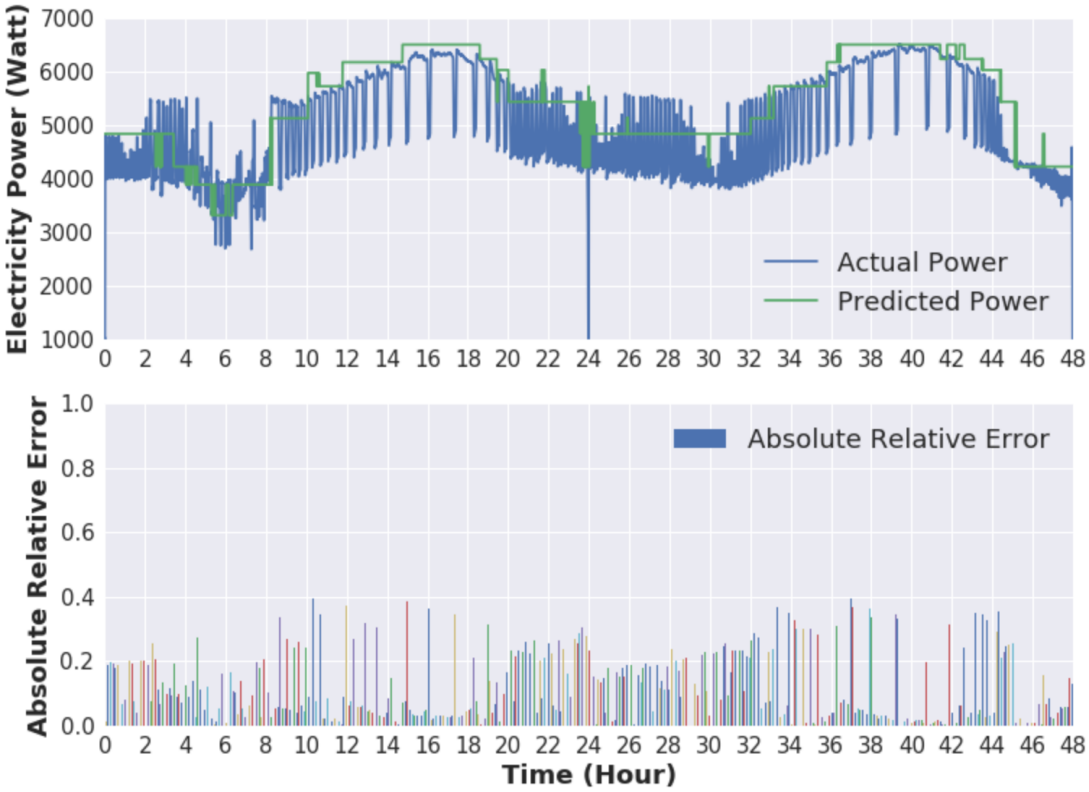
```
644.6792452124558
```

The best parameters are: {'max_depth': 2, 'n_estimators': 80}
The mean square error on test set is 644.6792452124558.
The absolute relative error for each model is shown below:

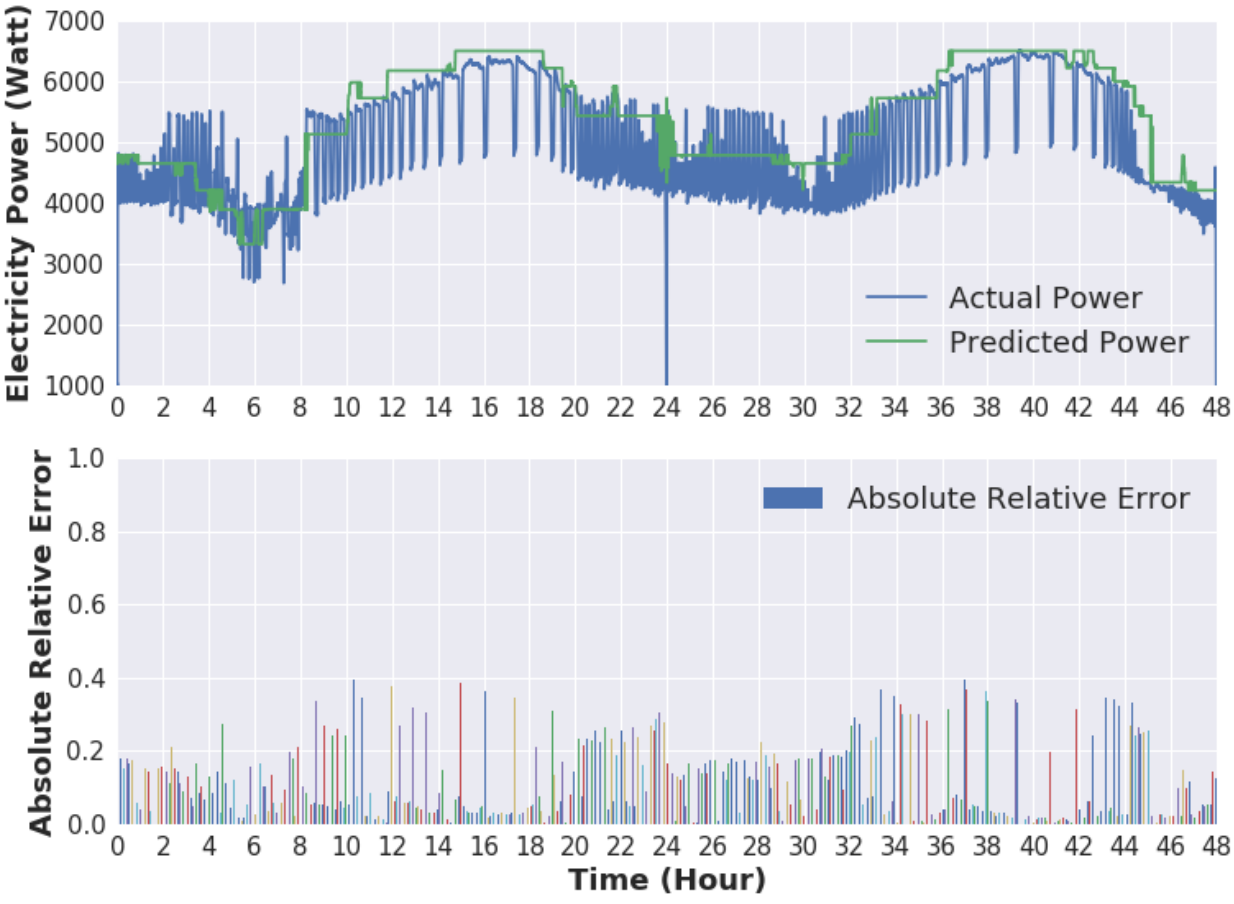
linear regression model (MLR)



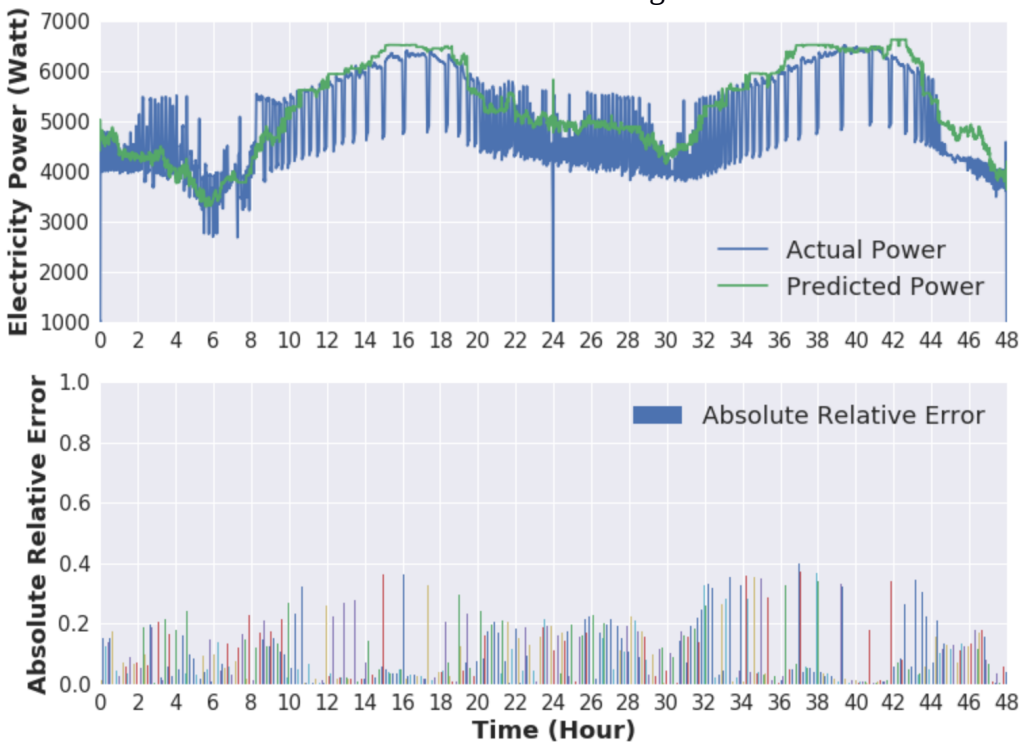
Decision Tree



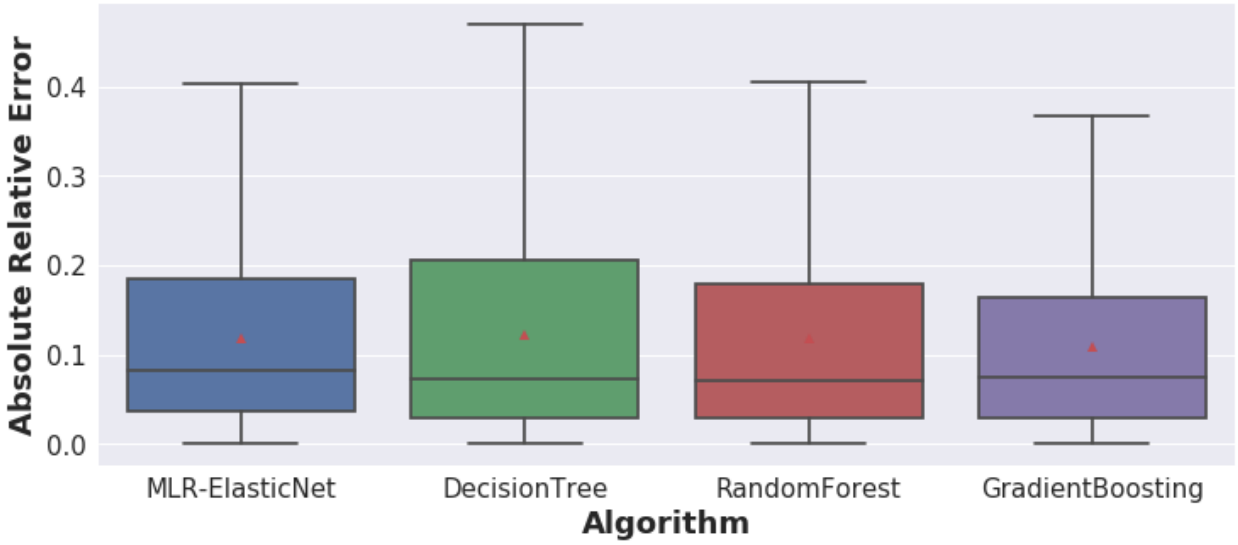
Random Forest



Gradient Boosting Tree



At last, we use boxplot to show the all four absolute relative error.



We see that the gradient boosting trees model performs the best based on absolute relative error.

Conclusion

In this project, we analysis the power consuming of air-conditioner with the environment of wind, solar, humidity and temperature. We tried to figure what's the relationship between these environment variables and the power consumption, and our result shows that temperature has the strongest relationship. Also, we use the regression models to forecast the power consumption within a day. We believe that our models presents the real situation well and the results could be helpful to save more energy in daily life. In the very end, we also get familiar with how to pre-process and present data in a intuitive way.

Reference

[1] U.S. Energy Information Administration. https://www.eia.gov/energyexplained/?page=us_energy_home#tab3
[2] State of the art in building modeling and energy performances prediction: A review
[3]Predicting electricity energy consumption: A comparison of regression analysis, decision tree and neural networks
[4] A review of artificial intelligence based building energy use prediction: Contrasting the capabilities of single and ensemble prediction models
[5] Multiple regression models for energy use in air-conditioned office buildings in different climates
[6] Using regression analysis to predict the future energy consumption of a supermarket in the UK
[7] Identifying key variables and interactions in statistical models of building energy consumption using regularization
[8] A regression-based approach to estimating retrofit savings
[9] A review and analysis of regression and machine learning models on commercial building electricity load forecasting
[10] Trees vs Neurons: Comparison between random forest and ANN for high-resolution prediction of building energy consumption
[11] Development of prediction models for next-day building energy
[12] Comparative assessment of low-complexity models to predict electricity consumption in an institutional building: Linear regression vs. fuzzy modeling vs. neural networks
[13] Regularization and variable selection via the elastic net
[14] Quinlan, J. Ross. C4. 5: programs for machine learning. Elsevier, 2014.
[15] Breiman, Leo. "Random forests." Machine learning 45.1 (2001): 5-32.
[16] Friedman, Jerome H. "Stochastic gradient boosting." *Computational Statistics & Data Analysis* 38.4 (2002): 367-378.