APPLIANCE ENERGY PREDICTION

Machine Learning Engineer Nanodegree

Capstone Project

Sarah M. Al-Abdulhadi

23 Jul 2018

I. Definition

Project Overview

This project aims to predict the energy consumption by home appliances. With the advent of smart homes and rising need for energy management, existing smart home systems can benefit from accurate prediction. If the energy usage can be predicted for every possible state of appliances, then device control can be optimized for energy savings as well. This is a case of Regression analysis which is part of Supervised Learning problem. Appliance energy usage is the target variable while sensor data and weather data are the features.

Dataset source: http://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction

Problem Statement

Develop a Supervised learning model using Regression algorithms to predict the appliance energy usage using sensor readings and weather data as features.

Metrics

Since this is a regression problem, the metric used will be ***"Coefficient of Determination"***, in other words denoted as R^2 (R squared) which gives a measure of the variance of target variable that can be explained using the given features.
It can be mathematically defined as:
Where,
SS_{res} = Residual sum of squares
SS_{tot} = Total sum of squares

For this project, I will use '***r2_score()***' function of the ***metrics*** module of scikit-learn library.

While "*Coefficient of Determination*" provides relative a measure of the how well the model fits the data, the ***RMSE (Root Mean Squared Error)*** gives absolute measure of how well model fits the data i.e. how close are the predicted values to the actual values.

Mathematically, RMSE can be defined as:

$$\sqrt{\frac{1}{N} * \sum_{i=1}^{N} (y_i - y'_i)^2}$$

where,
N = number of observations
y_i = Actual value of target variable
y_i' = Predicted value of target variable

In this project, I will calculate RMSE by calculating square root of ***mean_squared_error()*** function provided in the ***metrics*** module of scikit-learn library.

Therefore, the metrics to be used are:

I. R2 score
II. RMSE

These two metrics are helpful for this problem because of the following reasons:

I. It is a Regression based problem.
II. R2 score will show the statistical robustness of the model.
III. RMSE will give an idea about how accurate the predictions are to actual values.

II. Analysis

Data Exploration

The dataset has 28 features and 1 target variable described as follows:

NAME	DESCRIPTION	UNIT
Features		
T1	Kitchen Temperature	℃
T2	Living Room Temperature	℃
T3	Laundry Room Temperature	℃
T4	Office Temperature	℃
T5	Bathroom Temperature	℃
T6	Temperature outside Building (North)	℃
T7	Ironing Room Temperature	℃
T8	Teenager Room Temperature	℃
T9	Parents Room Temperature	℃
T_out	Outside Temperature (Weather Station)	℃
T_dewpoint	Dewpoint Temperature (Weather Station)	℃
RH_1	Kitchen Humidity	%
RH_2	Living Room Humidity	%
RH_3	Laundry Room Humidity	%
RH_4	Office Humidity	%
RH_5	Bathroom Humidity	%
RH_6	Humidity outside Building (North)	%
RH_7	Ironing Room Humidity	%
RH_8	Teenager Room Humidity	%
RH_9	Parents Room Humidity	%
RH_out	Outside Humidity (Weather Station)	%
Pressure	Outside Pressure (Weather Station)	mm Hg
Wind speed	Outside Windspeed (Weather Station)	m/s
Visibility	Visibility (Weather Station)	km
Date	Timestamp of the reading	yyyy-mm-dd HH:MM:SS
rv1	Random Variable 1	-
rv2	Random Variable2	-
Lights	Energy used by lights	Wh
Target Variable		
Appliances	Total energy used by Appliances	Wh

Out of these features, I won't be using the last 4 features as the problem is that of Regression and not Time series forecasting ("**Date**"). Also, the goal is to predict total energy consumption and not category-wise energy consumption ("**Lights**").
Therefore,
Number of features = 24
Number of target variables = 1
Number of instances in training data = 14,801
Number of instances in testing data = 4,934
Total number of instances = 19.735
Count of Null values = 0
All features have numerical values. There are no categorical or ordinal features in this dataset.

Descriptive statistics:

i Ranges of the columns:

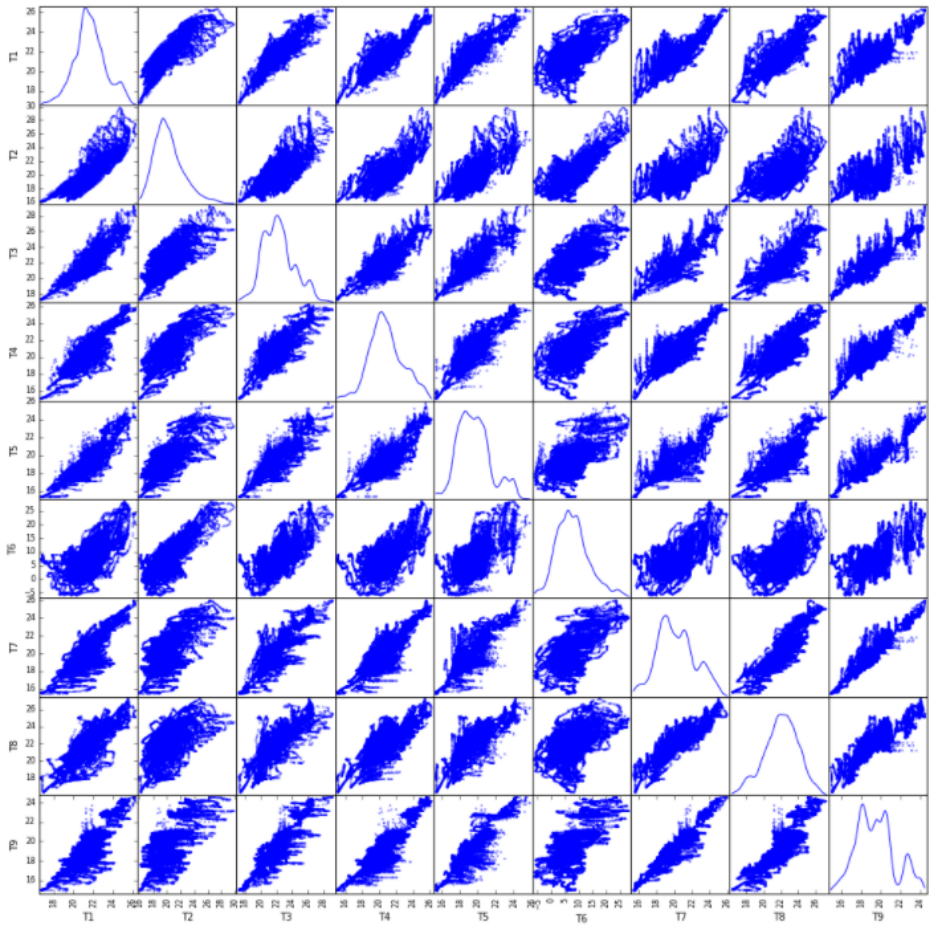
	T1	T2	T3	T4	T5	T6	T7	T8	T9
count	14801.000000	14801.000000	14801.000000	14801.000000	14801.000000	14801.000000	14801.000000	14801.000000	14801.000000
mean	21.691343	20.344518	22.278802	20.860393	19.604773	7.923216	20.273236	22.028122	19.493479
std	1.615790	2.202481	2.012934	2.048076	1.849641	6.117495	2.118416	1.960985	2.022560
min	16.790000	16.100000	17.200000	15.100000	15.340000	-6.065000	15.390000	16.306667	14.890000
25%	20.760000	18.790000	20.790000	19.533333	18.290000	3.626667	18.700000	20.790000	18.000000
50%	21.600000	20.000000	22.100000	20.666667	19.390000	7.300000	20.075000	22.111111	19.390000
75%	22.633333	21.500000	23.340000	22.100000	20.653889	11.226667	21.600000	23.390000	20.600000
max	26.260000	29.856667	29.236000	26.200000	25.795000	28.290000	26.000000	27.230000	24.500000

	RH_1	RH_2	RH_3	RH_4	RH_5	RH_6	RH_7	RH_8	RH_9
count	14801.000000	14801.000000	14801.000000	14801.000000	14801.000000	14801.000000	14801.000000	14801.000000	14801.000000
mean	40.267556	40.434363	39.243995	39.043799	51.014065	54.615000	35.410874	42.948244	41.556594
std	3.974692	4.052420	3.245701	4.333479	9.107390	31.160835	5.097243	5.210450	4.161295
min	27.023333	20.596667	28.766667	27.660000	29.815000	1.000000	23.260000	29.600000	29.166667
25%	37.363333	37.900000	36.900000	35.560000	45.433333	29.996667	31.500000	39.096667	38.530000
50%	39.693333	40.500000	38.560000	38.433333	49.096000	55.267500	34.900000	42.390000	40.900000
75%	43.066667	43.273453	41.730000	42.200000	53.773333	83.226667	39.000000	46.500000	44.326667
max	63.360000	54.766667	50.163333	51.090000	96.321667	99.900000	51.327778	58.780000	53.326667

	T_out	Tdewpoint	RH_out	Press_mm_hg	Windspeed	Visibility
count	14801.000000	14801.000000	14801.000000	14801.000000	14801.000000	14801.000000
mean	7.421836	3.782509	79.824197	755.480135	4.029001	38.290284
std	5.343737	4.194994	14.901776	7.389218	2.448171	11.789650
min	-5.000000	-6.600000	24.000000	729.300000	0.000000	1.000000
25%	3.666667	0.933333	70.500000	750.900000	2.000000	29.000000
50%	6.933333	3.483333	83.833333	756.000000	3.666667	40.000000
75%	10.433333	6.600000	91.666667	760.833333	5.500000	40.000000
max	26.100000	15.316667	100.000000	772.300000	14.000000	66.000000

	Appliances
count	14801.000000
mean	97.875144
std	102.314986
min	10.000000
25%	50.000000
50%	60.000000
75%	100.000000
max	1080.000000

ii. Scatter plots:
a. T1 to T9:



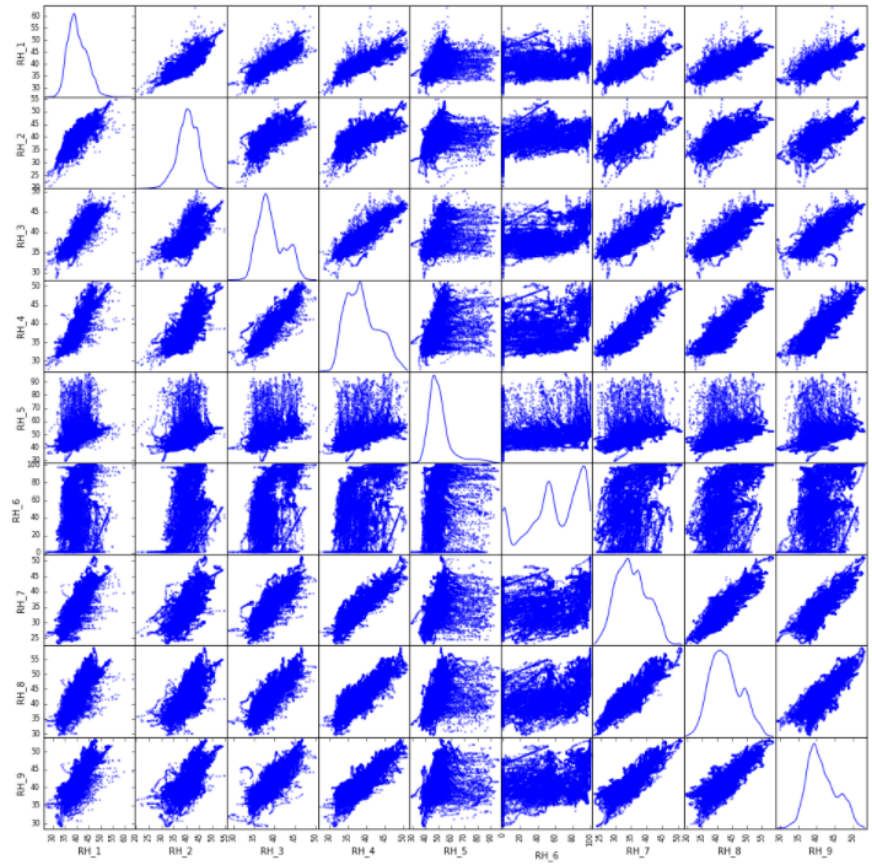
Some degree of correlation can be seen between T7 and T9. This can be confirmed by computing their Pearson coefficient which turns out as follows:

```
# Import pearson relation method from SciPy
from scipy.stats import pearsonr

# Calculate the coefficient and p-value
corr_coef, p_val = pearsonr(energy["T7"], energy["T9"])
print("Correlation coefficient : {}".format(corr_coef))
print("p-value : {}".format(p_val))
```

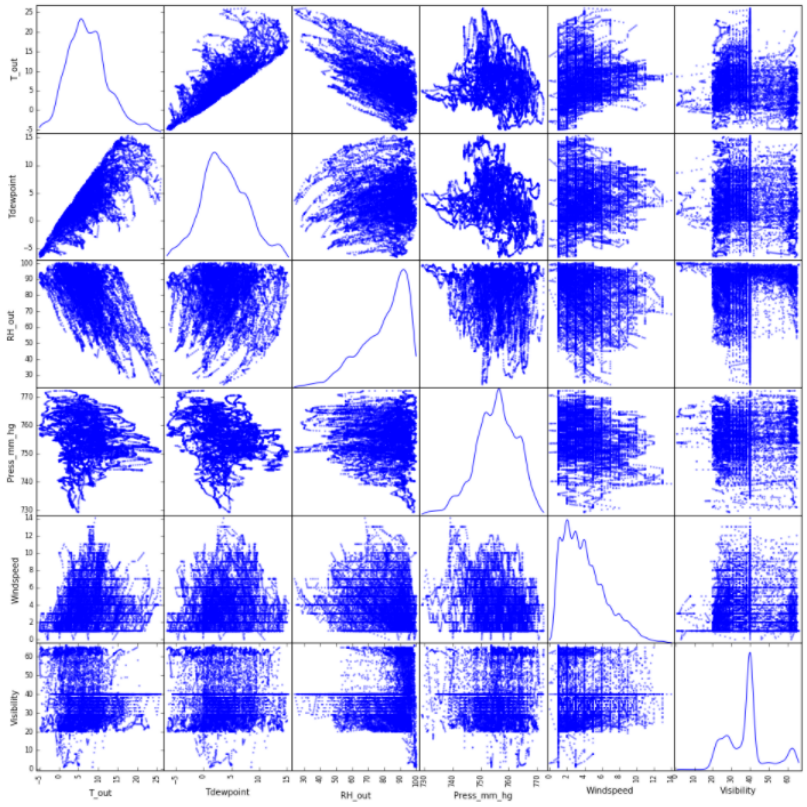
```
Correlation coefficient : 0.9460586115166221
p-value : 0.0
```

b. RH_1 to RH_9:

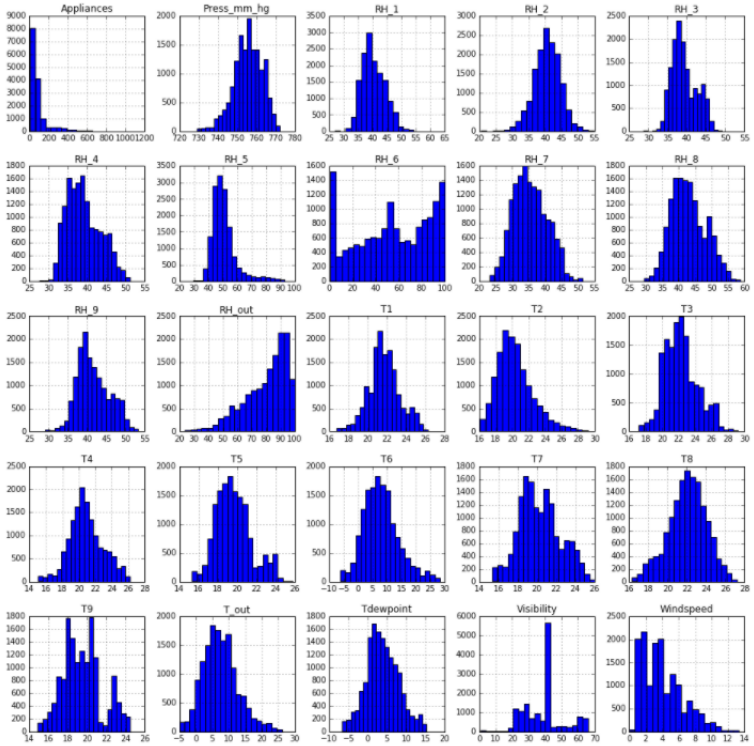


No significant correlation exists among different humidity values and weather among weather parameters like Pressure, Windspeed, Temperature, etc. which can be confirmed from the plots b. And c.

c. Weather data:



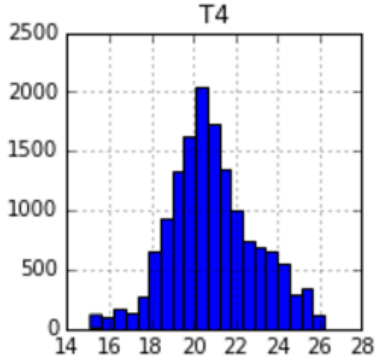
i iii. Distribution of all the columns: From



From this plot, it can be concluded that no columns have a distribution like the **Appliances** column, which is our target variable. Therefore, we can deny a linear relationship of any single feature independently with the target variable.

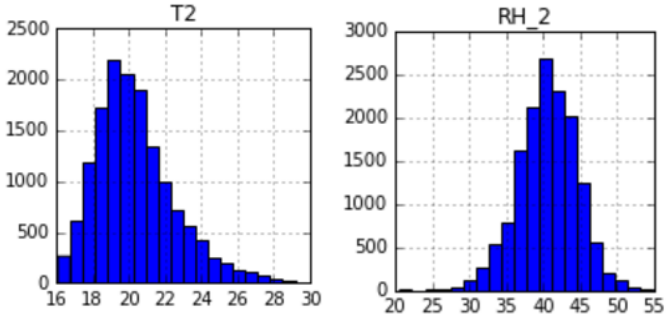
Exploratory Visualization

i. Most features have their values in normal distribution. For example:

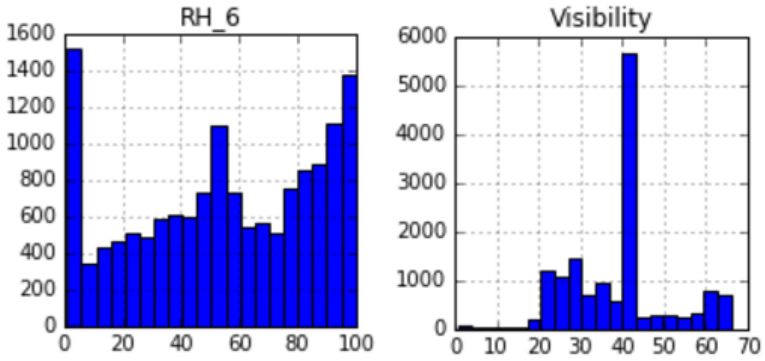


ii. Out of which, some features are skewed left/right as shown below:

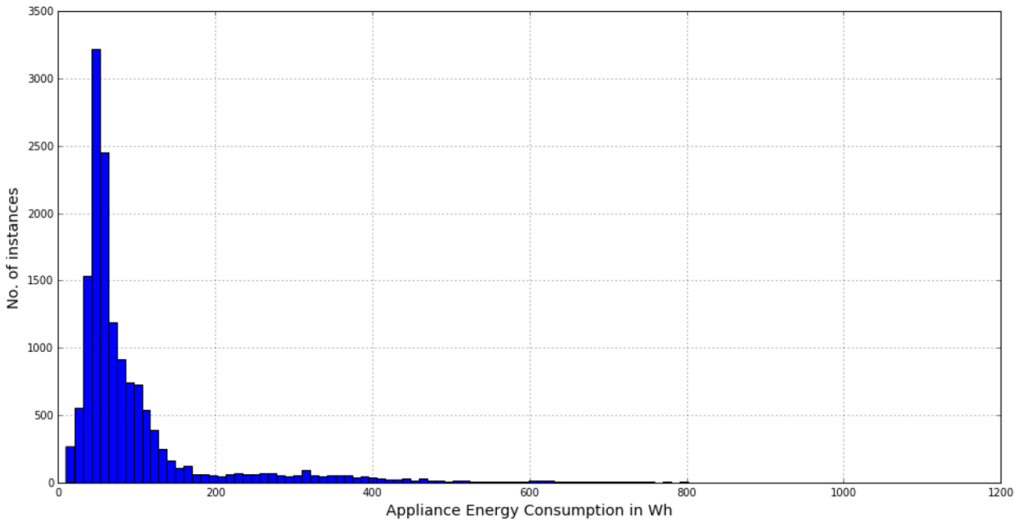
Left Right



iii. Some features don't have normal distribution as shown below:



Distribution of the target variable:



Observations:

i. Most features are normally distributed.
ii. The target variable has a highly skewed distribution and it doesn't have linear relation with any other features.
iii. The feature ***T9*** is highly correlated with features ***T3***, ***T5*** and ***T7***.
iv. The feature ***T6*** is highly correlated with feature ***T_out***.

Algorithms and Techniques

I will try the following algorithms for Regression:

The most basic Regression algorithm is Linear Regression. If a Linear model can explain the data well, there is no need for further complexity. As modification to original Least Squares Regression, we can apply Regularization techniques to penalize the coefficient values of the features, since higher values generally tend towards overfitting and loss of generalization. Regularization techniques enhance performance of Linear models greatly. Also, there very few practical cases when a Linear model can fit the data well without Regularization.
In case of Regularization, depending upon whether we add the absolute values of coefficients or their squares to our loss function, the problem of Linear Regression is transformed into Lasso or Ridge Regression respectively.

i. Linear Models :

1. Linear Regression
2. Ridge Regression
3. Lasso Regression

The next category of algorithms is of Tree based Regression models. An important advantage of Tree based models is that they are robust to outliers compared to Linear models. We haven't seen that a Linear relationship between any feature and the target variable, it is likely that Regression tress will turn out to be better than Linear models.

Given the substantial number of features, it is evident that a Decision Tree will overfit the data. Hence, I have skipped it and directly jumped towards ***ensemble*** methods listed below, which include building multiple regressors on copies of same training data and combining their output either through mean, median, mode (***Bagging***) or growing trees sequentially (i.e. each tree is built from data of the previous tree) and using weighted average of these weak learners (a learner which performs just a little better than chance (50%)) (***Boosting***).

Random Forests is one of the primary ***Bagging*** methods and works well on high dimensional data like ours. Extreme Trees Regression goes one step further by making splits Random. Gradient Boosting Machines is a type of Boosting method. It builds an additive model in a way that performance always increases.

ii. Tree based models:

1. Random Forests
2. Gradient Boosting Machines
3. Extremely Randomized Trees

Finally, one of the primary algorithms for non-linear hypothesis is a neural network. Neural networks work great when there is a complex nonlinear relationship between the inputs and the output. Although they generally have superior performance, one of their downside is that they take very long time to train. I will be using a Multi-Layer Perceptron as my choice of Neural network. The error function is squared

iii. Neural Networks:

1. Multi-layer Perceptron

Benchmark

The benchmark model is Linear Regression on unscaled data using all the features.

Observations:

i. R2 score on training data: 14.687%

ii. R2 score on test data: 14.258%

iii. RMSE on test data = 0.926 (For calculating RMSE, the data was scaled so that comparison with other models is easier)

iv. Time taken to fit: 0.032 seconds

III. Methodology

Data Preprocessing

Ranges of features irrespective of units

Ranges of features irrespective of units Temperature	-6 to 30
Humidity	1 to 100
Windspeed	0 to 14
Visibility	1 to 66
Pressure	729 to 772
Appliance Energy Usage	10 to 1080

Due to different ranges of features, it is possible that some features will dominate the Regression algorithm. To avoid this situation, all features need to be scaled.

Thus, the data was scaled to 0 mean and unit variance using the ***StandardScaler*** class in ***sklearn.preprocessing*** module.

The first row of data is shown before and after scaling.

Before scaling:

T1	20.200000
RH_1	37.500000
T2	17.823333
RH_2	39.300000
T3	20.290000
RH_3	36.560000
T4	18.200000
RH_4	37.290000
T5	17.926667
RH_5	47.633333
RH_6	67.666667
T7	18.463333
RH_7	29.390000
T8	21.390000
RH_8	35.663333
RH_9	35.500000
T_out	2.800000
Press_mm_hg	744.000000
RH_out	86.666667
Windspeed	2.666667
Visibility	28.000000
Tdewpoint	0.766667
Appliances	70.000000

After scaling:

T1	-0.923012
RH_1	-0.696318
T2	-1.144741
RH_2	-0.279932
T3	-0.988045
RH_3	-0.826966
T4	-1.299016
RH_4	-0.404723
T5	-0.907291
RH_5	-0.371220
RH_6	0.418863
T7	-0.854395
RH_7	-1.181242
T8	-0.325420
RH_8	-1.398182
RH_9	-1.455508
T_out	-0.864936
Press_mm_hg	-1.553686
RH_out	0.459187
Windspeed	-0.556489
Visibility	-0.872853
Tdewpoint	-0.718939
Appliances	-0.272454

I also removed the columns ***T6*** and ***T9*** which had a significant correlation with columns ***T_out*** and (***T_3, T_5, T_7***) respectively. As a result, there are 22 features in the training data.

Implementation

The model implementation is done in 3 steps:

i. Create a ***pipeline()*** function to execute each Regressor and record the metrics.
ii. Pass each Regressor to above pipeline function from ***execute_pipeline()***.
iii. Consolidate the obtained metrics into a DataFrame in the function ***get_properties()*** and plot these metrics using a bar graph.

List of Algorithms tested:

i. sklearn.linear_model.Ridge
ii. sklearn.linear_model.Lasso
iii. sklearn.ensemble.RandomForestRegressor
iv. sklearn.ensemble.GradientBoostingRegressor
v. sklearn.ensemble.ExtraTreesRegressor
vi. sklearn.neural_network.MLPRegressor

Performance metric used:
R2 score (the ***r2_score()*** method mentioned in section 1.3) which is internally used by the ***score()*** method of all Regressors mentioned above.
RMSE will be calculated by taking square root of MSE value calculated using ***mean_square_error()*** function.

Results:

	RMSE	Testing scores	Training scores	Training times
Ridge	0.936121	0.123677	0.137409	0.0260139
Lasso	1	0	0	0.0315361
RandomForestRegressor	0.728899	0.468707	0.91342	12.097
GradientBoostingRegressor	0.86821	0.246212	0.331539	6.69755
ExtraTreesRegressor	0.664811	0.558027	1	3.38832
MLPRegressor	0.844788	0.286334	0.353317	17.8957
Linear Regression (Benchmark)	0.926026	0.142476	0.146873	0.0375407

As observed from results, ***ExtraTreesRegressor*** performs better than all other regressors in terms of all metrics except for Training time where Linear models outperform it. Even then, it's training time is less than all the other Regressors.

Refinement

For refining the model, I tweaked the following properties of ExtraTreesRegressor:

i. n_estimators: The number of trees to be used.
ii. max_features: The number of features to be considered at each split.
iii. max_depth: The maximum depth of the tree.

My feature superset is as follows:

```
param_grid = {
    "n_estimators": [10, 50, 100, 200, 250],
    "max_features": ["auto", "sqrt", "log2"],
    "max_depth": [None, 10, 50, 100, 200, 500]
}
```

Here, the values of ***max_features*** parameter defines the function to be applied on total number of features to obtain the new number of features to be considered during splits.

For ***max_depth***, None means keep splitting until all leaves are pure or they have less samples than ***min_samples_split*** parameter whose default value is 2.

Before Tuning, the R2 score on test set was 0.558. After tuning, it rose to 0.610, a performance gain of **5.2%**.

A summary of challenges faced an overcame:

1. Always check for correlated features in a high dimensional dataset, and remove redundant features with high correlation.
2. Feature scaling is a must for Regression.
3. Use a seed generator for reproducible results.
4. If you want to maintain separate copies of DataFrames with scaled data, it is viable to create dummies using original DataFrame's index and columns and then filling it with scaled data.
5. The pipeline should be as modular as possible. For example, I can easily add an algorithm to the list of algorithms to be tested in the ***execute_pipeline()*** function without changing the model implementation function ***pipeline()***.
6. It is easier to plot various properties of the models if they are consolidated into a DataFrame rather than storing and manually plotting them individually.
7. Cross validation is very useful for finding out the best model.
8. For performing Exhaustive search or Random search in the hyperparameter space for tuning the model, always parallelize the process since there are a lot of models with different configurations to be fitted. (Set ***n_jobs*** parameter with the value -1 to utilize all CPUs)
9. One effective way to check the robustness of the model is to fit it on a reduced feature space in case of high dimensional data. Select the first '***k***' (usually >= 3) key features for this task.

IV. Results

Model Evaluation and Validation

Features of the untuned model:

i. n_estimators = 10
ii. max_features = n_features = 22
iii. max_depth = None

Features of best model after hyper parameter tuning:

i. n_estimators = 250
ii. max_features = log2(n_features) = log2(22) ~ 4
iii. max_depth = None

Robustness check:

The best model is trained on reduced feature space having only 5 highest ranked features in terms of importance instead of 22 features.

R2 score on test data = **0.499**.
R2 score of untuned model = 0.558.
Difference = 0.059 or 5.9%.

RMSE on test data = 0.708
RMSE of untuned model = **0.665**
Difference = 0.343

Therefore, we can see that even though the feature space is reduced drastically (by more than 75%), the relative loss in performance on test data is less.

Justification

Parameters/Models	Final Model	Benchmark Model	Difference
Training R2 score	1.0	0.147	0.853
Testing R2 score	0.61	0.142	0.468
RMSE on test data	0.624	0.926	0.302

Based on the improvements recorded above, the final tuned model can be deemed as a satisfactory solution.

V. Conclusion

Free-Form Visualization

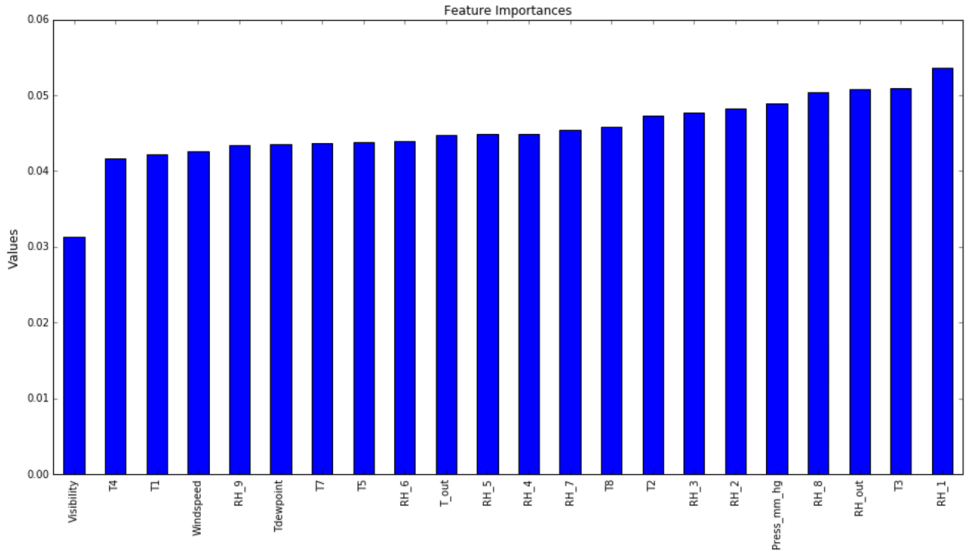
According the best fitted model, the feature importance is as follows:

```
Most important feature = RH_1
Least important feature = Visibility

Top 5 most important features:-
RH_1
T3
RH_out
RH_8
Press_mm_hg

Top 5 least important features:-
Visibility
T4
T1
Windspeed
RH_9
```

Visual representation of all features:



It can be observed that on an average, humidity affects power consumption more than temperature. This is evident from the fact that more number of humidity readings are towards the higher end of the graph as compared to temperature readings.

Also, out of weather parameters, Humidity and Atmospheric pressure affect power consumption more significantly than others. This is in line with the general assumption that factors like Windspeed and Visibility shouldn't affect the power consumption inside the home.

An important conclusion drawn from this visualization is that that although natural humidity cannot be controlled, controlling humidity inside the home can lead to energy savings.

Reflection

This project can be summarized as the sequence of following steps:

1. Searching for a problem by looking at datasets on UCI Machine Learning repository and Kaggle and deciding between Classification and Regression problems.
2. Visualizing various aspects of dataset.
3. Preprocessing the data and feature selection.
4. Deciding the algorithms to be used to solve the problem.
5. Creating a benchmark model.
6. Applying selected algorithms and visualizing the results.
7. Hyper parameter tuning for the best algorithm and reporting the test score of best model.
8. Discuss importance of selected features and check the robustness of model.

Out of this, I found steps 1, 2 and 5 very interesting. Deciding between Classification and Regression was an important hurdle. But, as I had approached a few classification problems before, I decided that it would be more exciting to solve a Regression based problem.

Therefore, visualizing a dataset from the point of view of solving a Regression problem where your output isn't defined among a few classes was particularly challenging.

Also, in the case of Classification, a benchmark model can be created using the concept of chance i.e. Accuracy = 1/n_classes. In this project, I had initially decided to create two benchmark models, one that would always return the mean of the target variable and one which would return the median. But, after visualizing the data and concluding that there are no Linear relationships of any feature with the target variable, I realized that a Linear Regression model may serve as a better benchmark.

Improvement

A few of the ways the solution can be improved are:

i. Discarding seemingly irrelevant weather features like Windspeed and Visibility.
ii. Performing more aggressive feature engineering.
iii. Using Grid search instead randomized search to search the parameter space exhaustively and determine the best solution.
iv. As an add-on to previous step, more number of parameters can be added to the parameter space.