Practical 1: Predicting energy performance of residential buildings

180028981

March 2019

Contents

1 Introduction

The main aim of this assignment is to train a regression model in order to predict the heating and cooling load of a house given a certain number of parameters. An additional aim of this practical is to thoroughly document the process of the creation of this algorithm from loading and cleaning the data for the first time until the evaluation of the final results. The report will focus on the following sections:

- Loading and Cleaning the Data
- Analysing and Visualising the Data
- Preparing the Inputs and Choosing a suitable Subset of Features
- Selecting and Training a Regression Model
- Evaluating the Performance of the Model

The model will be trained using the scikit-learn library(Version 0.20.3).

2 Loading and Cleaning the Data

This was done using the python library pandas (Version 0.24.1). By using its `read_csv()` function i was able to save the data to a pandas DataFrame as shown below.

```
file = pd.read_csv(file_path)
column_number = len(file.columns)
columns_used = [0, 1, 2, 3, 4, 5, 6, 7]
training = pd.read_csv(file_path, usecols=columns_used, index_col=None)
results = pd.read_csv(file_path, usecols=[column_number - 2, column_number - 1],
                        skiprows=None, index_col=None)
```

I specify which columns are going to be used for the training data and then i save the last two columns in separate DataFrame called results.

3 Analysing and Visualising the Data

In order to gain a better understanding over the data, we are going to try and visualise the relation between the features and the expected output. This was done by using the matplotlib library (Version 3.0.3). Below is the code implementation

```python
def plot_graphs(training, results):

    x_column_names = training.columns
    y_column_names = results.columns

    x_features_num = training.shape[1]
    y_features_num = results.shape[1]

    x = training.values
    y = results.values

    for y_col in range(0, y_features_num):
        plt.figure(num=y_column_names[y_col]+"␣Values")
        for x_col in range(0, x_features_num):
            plt.subplot(x_features_num/2, 2, x_col+1).set_title(str(y_column_names[
                y_col]) + "␣values␣for␣" +
                                                            str(x_column_names[x_col]))
            plt.scatter(x[:, x_col], y[:, y_col])
            plt.tight_layout(pad=0.3, w_pad=0.5, h_pad=0.4)
        plt.show()
        plt.clf())
```

There are two passing parameters for this method which are the training data and expected results which were acquired in the previous section. This method loops through each individual training column and plots that column as a function of the expected output column. Since we are currently using 8 features and there are 2 output columns, below are the 16 graphs that were generated.

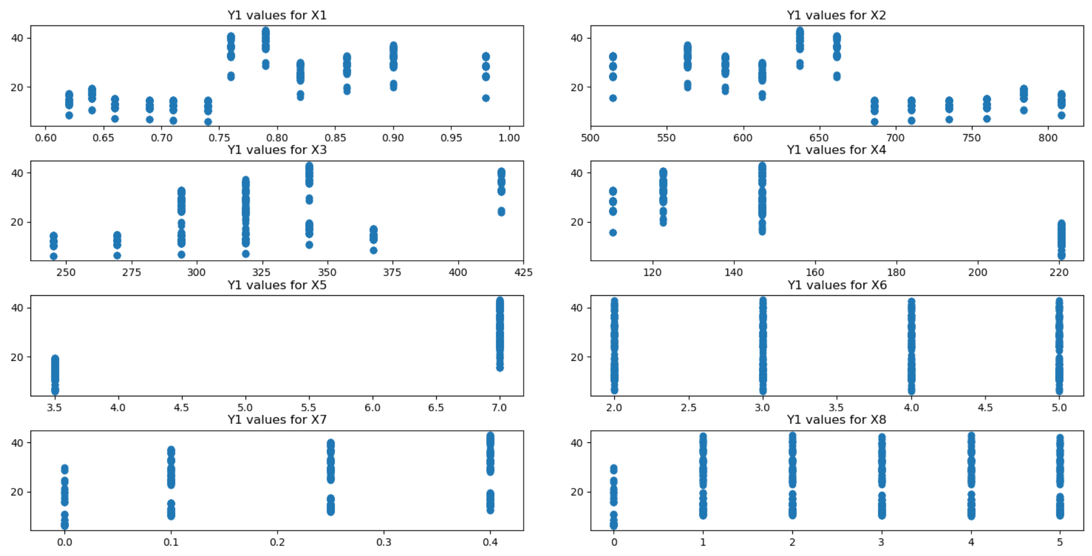

Figure 1: Y1 Values for X

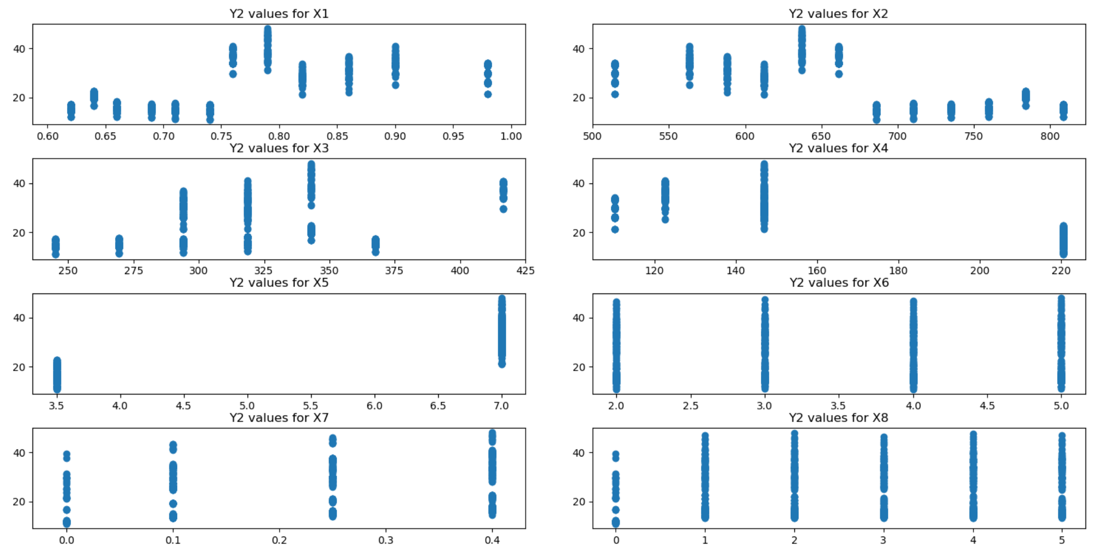

Figure 2: Y2 Values for X

4 Preparing the Inputs and Choosing Features

In order to better evaluate which subset of the current features is the most optimal to use, we will be using the Pearson's Correlation Coefficient (PCC). PCC is a measure of the linear correlation between two variables and we can use it to see how much does a change in variable x affect variable y. The values of PCC can vary from -1 to 1 with -1 signifying a direct correlation and -1 signifying a total negative correlation. The closer the value is to 0, the less the correlation between the two values. By using the statistic library of SciPy (Version 1.2.1). This library has a function that calculates PCC by the use of a single method like seen below.

```python
def pearson_plot(training, results):
    x_column_names = training.columns
    y_column_names = results.columns

    x_features_num = training.shape[1]
    y_features_num = results.shape[1]
    plt.figure(num="Pearson's␣Correlation␣coefficient␣for␣Y␣Values")

    for y_col in range(0, y_features_num):
        pp = []
        for x_col in range(0, x_features_num):
            value = pearsonr(training.values[:, x_col], results.values[:, y_col])[0]
            pp.append(value)

        plt.subplot(1, 2, y_col+1).set_title("PCC␣for␣" + str(y_column_names[y_col]))
        plt.bar(x_column_names, pp)

    plt.show()
```

This method calculates the PCC for each of the two output columns in regards to each feature and plots the results using a bar chart from matplotlib. Below are the results.

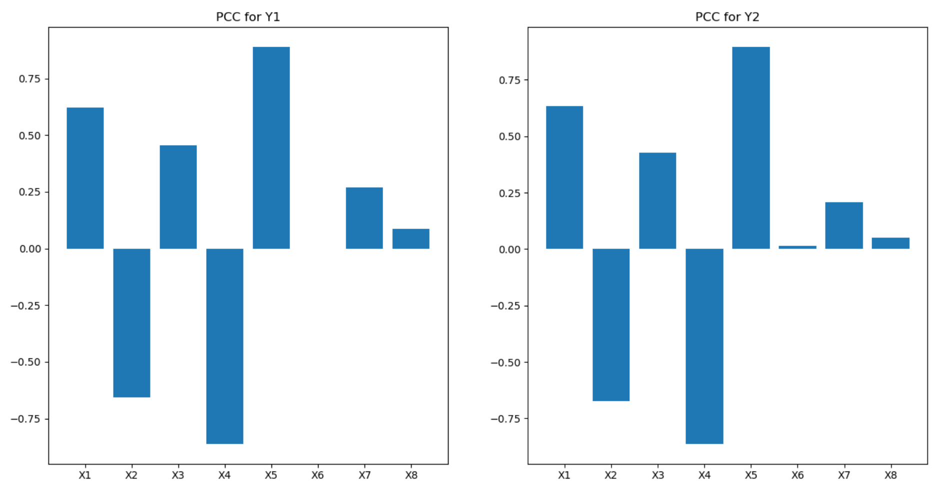

Figure 3: Y2 Values for X

As we can clearly see from the graph in Figure 3, features X6 and X8 have a value of PCC which is very close to 0. This implies that they have little to no correlation with the final output of the program. In order to minimize computation cost for our algorithm we shall remove them from our training set.

```
    training = training.drop(columns=['X6', 'X8'])
```

Now that a theoretically appropriate subset of features has been selected, the data needs to be scaled in order to improve the scalability of the model. I decided to use feature scaling instead of standard score since the data is not symmetric. To normalize the data, their values will be set in a range from 0 to 1 and the way this was implemented was using the MinMaxScaler from the scikit-learn library as shown below:

```
    scaler = MinMaxScaler()
    scaler.fit(training)
    training = scaler.transform(training)
    scaler.fit(results)
    results = scaler.transform(results)
```

This turns our training set from this:

```
Training Data:
       X1     X2     X3      X4   X5   X7
0    0.98  514.5  294.0  110.25  7.0  0.0
1    0.98  514.5  294.0  110.25  7.0  0.0
2    0.98  514.5  294.0  110.25  7.0  0.0
3    0.98  514.5  294.0  110.25  7.0  0.0
...
```

to this:

```
Training Data:
[[1.        0.        0.28571429 0.        1.        0.        ]
 [1.        0.        0.28571429 0.        1.        0.        ]
 [1.        0.        0.28571429 0.        1.        0.        ]
 ...]
```

Final step before we move onto the model creation is splitting our data into a training set, a validation set and a testing set. This was done by using the method from the scikit-learn library called `train_test_split()`. This method splits the data into two 4 sub lists and it was implemented as shown below:

```
X_train, X_testing, y_train, y_testing = train_test_split(training, results,
                                   test_size=0.10,shuffle=False,random_state=40)
X_train, X_validation, y_train, y_validation = train_test_split(X_train, y_train,
                                         test_size=0.1111111111, shuffle=True)
```

Firstly, we split the code into a training and testing set. Lists which have an 'X' in their variable name (e.g. X_train) contain the feature values from the initial data. Lists which have an 'Y' in their variable name (e.g. Y_train) contain the output values from the initial data. The size of the testing set is set to 10% of the overall data set. The random_state parameter is set to 40 and shuffle is set to False so that testing set will remain the same regardless of how many times the program is run. This is to avoid overfitting. Afterwards, the training set is split once more to the final training set and a validation set. Here the parameter shuffle is set to true in order to randomize the data each time and gain a more objective final result. The test_size parameter is set to 0.1111111111 in order for the validation and testing set to have the same size (i.e. 10% of the overall data set).

5 Linear Regression

In order to implement the Linear Regression model the sckit-learn library was used as follows:

```
def get_lr_classifier(data, results):
    lr = LinearRegression()
    lr.fit(data, results)
    return lr
```

The classifier was created and fitted with the data. After that it was ready for testing. The mean squared error was calculated:

```
    error = cross_val_score(linear_classifier, X_validation, y_validation, cv=2,
        scoring="neg_mean_squared_error").mean()
```

The cross validation variable was set to 2 because after some testing it was outputting better results than the default value of 3. A method was implemented to calculate the Coefficient of determination r^2 given a model:

```
def score_classifier(classifier, data, results):
    print(classifier.score(data, results))
```

Finally, a method was created to plot the results of the model (line graph) pitted against the expected output (scatter plot). The method is below:

```
def plot_lr_results(model, X, y):
    theta = model.coef_
    y_predicted = model.predict(X)

    x_features_num = X.shape[1]
    g = np.empty((len(X), 2))

    for i in range(0, len(X)):
        gUnitY1 = 0
        gUnitY2 = 0
        for j in range(0, x_features_num):
            gUnitY1 += theta[0][j]*X[i][j]
            gUnitY2 += theta[1][j]*X[i][j]

        g[i][0] = gUnitY1
        g[i][1] = gUnitY2

    plt.subplot(1, 2, 1).set_title("Y1")
    plt.scatter(g[:, 0], y[:, 0], c='r')
    plt.plot(g[:, 0], y_predicted[:, 0], c='b')
    plt.subplot(1, 2, 2).set_title("Y2")
    plt.scatter(g[:, 1], y[:, 1], c='r')
    plt.plot(g[:, 1], y_predicted[:, 1], c='b')
    plt.show()
    plt.clf()
```

This method calculates a single value using all the values of a row of features. This is used by using the theta variable which is obtained by the line `theta = model.coef_` and the formula used to combine the values is below:

$$= g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

Figure 4: Theta Values

6 Random Forest Regression

Random Forest regression is again implemented using the sckit-learn library as shown below:

```
def get_rf_classifier(data, results):
    rf = RandomForestRegressor()
    rf.fit(data, results)
    return rf
```

The same methods as before were used to plot and get the mean squared error as well as the coefficient of determination.

7 Evaluation

7.1 Linear Regression

Below are 5 different results from the linear regression model. The results include the Coefficient of determination r^2 and the squared mean error of the testing and validation set. Below are two

	Score Classifier:	SME Validation Set	SME Testing Set:
1	0.9312164433957616	0.010444518877392159	0.012846024369907831
2	0.9341629379941497	0.00507167095217705	0.012846024369907831
3	0.9350986926202294	0.009065419781342704	0.012846024369907831
4	0.9342832245838785	0.01002480427020241	0.012846024369907831
5	0.9296461526260524	0.010269806431870511	0.012846024369907831

Table 1: Linear Regression Testing Results

graphs created by the plot_lr_results() method which was mentioned in Section 5. The first graph shows the results when the validation set was used and the second one shows the results when the testing set was used.

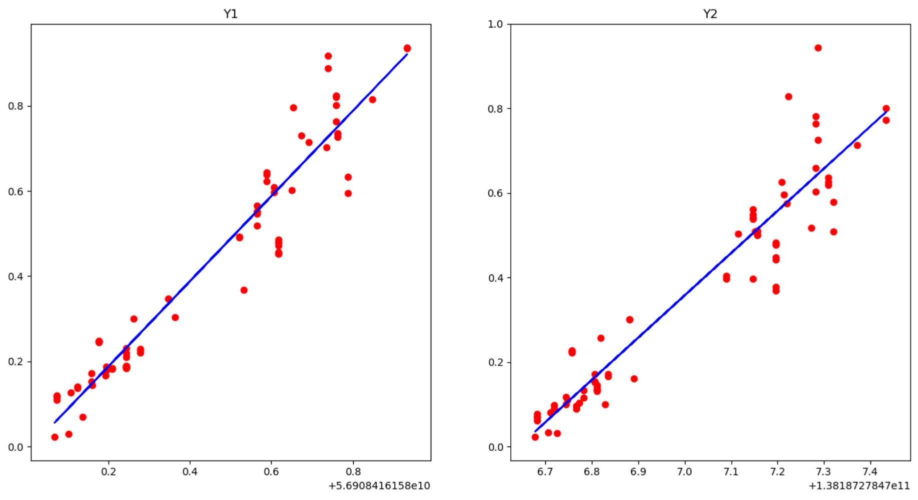

Figure 5: Results of the linear regression model using the validation set

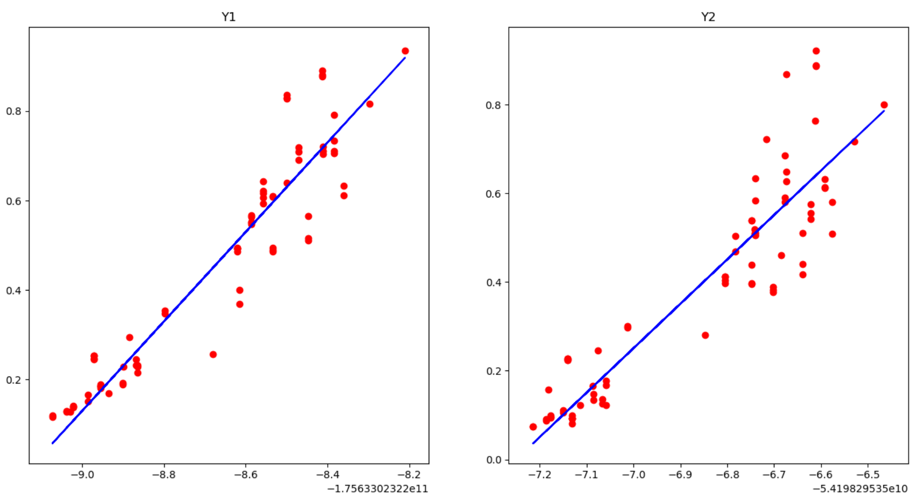

Figure 6: Results of the linear regression model using the testing set

7.1.1 Random Forest Regression

Below are the results of testing the Random Forest Regression model

	Score Classifier:	SME Validation Set	SME Testing Set:
1	0.9858587481313416	0.005665349751918082	0.004998918849458053
2	0.9863202515677302	0.005262687209348091	0.005641335990303166
3	0.9865656940373652	0.00932919861085223	0.006011259205205909
4	0.985587201451884	0.003975307495263931	0.003961485110756021
5	0.9860716126002897	0.0033830593997991078	0.0034561816237220567

Table 2: Random Forest Regression Testing Results

7.1.2 Ending Notes

As we can observe from the tables, the random forest regression model outperforms the linear regression model substantially as mentioned in the paper. Unfortunately, due to time constraints no further evaluation could be done to the algorithms. This was an enjoyable project and i believe that i focused more on the steps leading up to the implementation of the algorithms rather than the actual implementation and testing. This was due to time constraints and hopefully the next time i receive an open ended project such as this the opportunity to explore will not go wasted.

8 References

- VanderPlas J. (2016), Python Data Science Handbook, 1st Edition, O'Reilly Media
- Machine Learning Lecture Notes from St. Andrews University, written by Terzić K. and Harris-Birtill D.
- scikit-learn Version 0.20.3