Machine Learning Assignment 3

rESULTS AND report

Emmett Fitzharris R00222357

2024

# Task 1

In Task 1 I begin by separating the labels and targets into separate data frames. Our targets, “Heating load” and “Cooling Load” are the last two columns in the data, so I use a slice to select the appropriate information.

A screen shot of a computer

Description automatically generated

To verify my data is correct, I print a readable list of Labels and Targets to console.

A screenshot of a computer

Description automatically generated

The maximum and minimum of each target are calculated simply using the max() and min() methods.

A black screen with white text

Description automatically generated

One again I print the result, this time creating a merged data frame to make print formatting of the table simple.

A black background with white text

Description automatically generated

# Task 2

Task 2 does not have immediate output. The result will be used in later tasks.

First, I create a function which calculates the value of a polynomial model. This accepts three parameters:

* “degree”
  + Complexity of the model.
* “feature\_vectors”
  + Input data
* “coefficients”
  + The weights assigned to each term.

A screen shot of a computer code

Description automatically generated

I then create a function which calculates the number of parameters which are required for a given degree. The degree decides the size of the coefficient vector, and therefore the complexity of the model.

I use a nested loop to iterate through all possible combinations of exponents for the polynomial terms, to find cases where the sum is less than the degree. The number of valid results is the number of parameters we should use.

A computer screen shot of a number of text

Description automatically generated with medium confidence

# Task 3

In task three I create a function which calculates the value of the model function and it’s Jacobian at a given linearization point.

The “polynomial\_model()” function from task 2 is called on line 88 to calculates the estimated target vector with initial coefficients, then again in the loop on line 103 with slightly modified coefficients to compute the partial derivatives for the Jacobian.

A screen shot of a computer program

Description automatically generated

# Task 4

In task 4 I create the function “calculate\_optimal\_parameter\_update()”.

This function creates a normal matrix, by performing matrix multiplication of the transposed Jacobian and a regular matrix.

Residuals are then calculated as the difference between training and estimated target values, and are used to find the product of the transpose of the Jacobian and the residuals, as the rhs (“Right hand side”).

A screen shot of a computer program

Description automatically generated

The rhs is used to solve the linear matrix equation, which provides the optimal parameter update.

# Task 5

In task 5 I create the function “fit\_polynomial\_model()”.

In this function I start by Initializing an array of zeros matching the number of parameters.

I then create a loop, which repeatedly calculates coefficient parameter updates using the function defined in task 5, to fit the model to the data.

When the change is small enough that it is not significant, I break out of the loop.

A screen shot of a computer

Description automatically generated

# Task 6

In task 6 I create 2 primary functions, “cross\_validate\_polynomial\_model()” and “evaluate\_polynomial\_degrees()”, then call them from the “task6()” function which itself is called in main().

A screenshot of a computer program

Description automatically generated

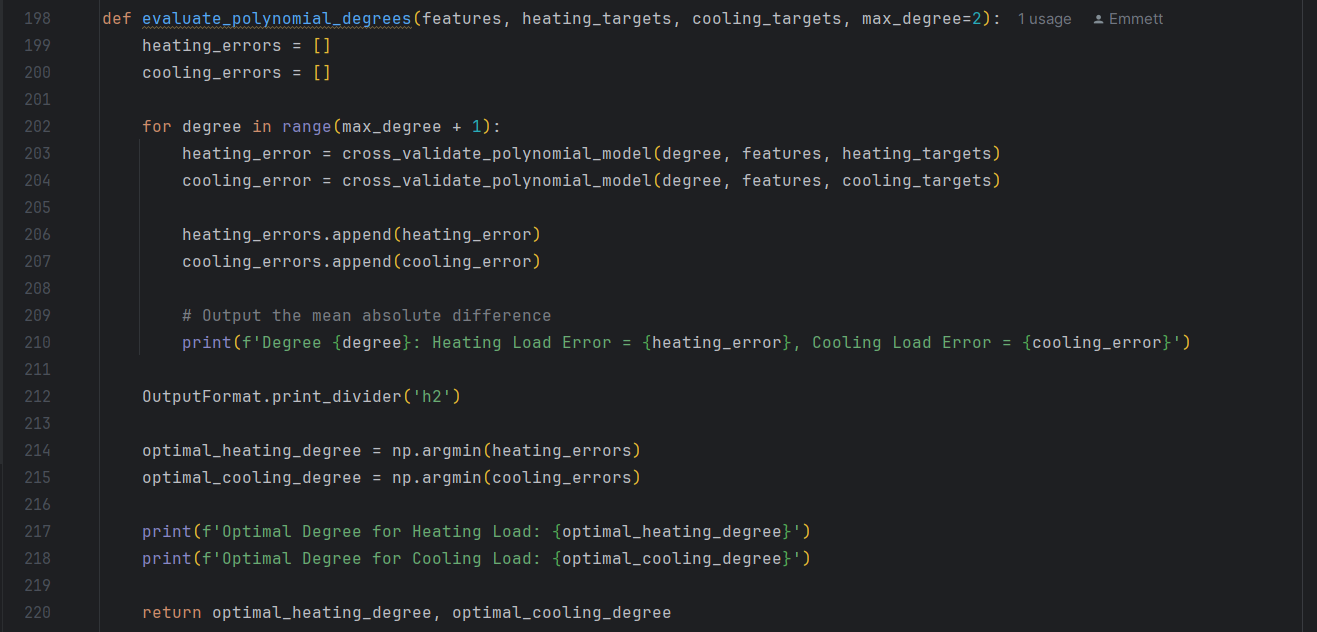
cross\_validate\_polynomial\_model() accepts degree, features and targets, and performs a cross fold validation. I first split the data into 5 folds, and arbitrary number which can be controlled by the parameter for further tuning.

Each fold is split into test and training data, then the model is fit to the training data and evaluated based on the test data. They then return the mean differences.

A screen shot of a computer code

Description automatically generated

evaluate\_polynomial\_degrees() calls the above for each target set. Then outputs the mean difference between expected values for each target (Heating and Cooling).



Finally using the numpy argmin() function, we find the index of the minimum value of the error arrays. This corresponds to the degree of the polynomial which showed the lowest error.

This value is output for each target.

A screenshot of a computer program

Description automatically generated

The optimal value for degree in both cases is 2, as the error is at its lowest.

# Task 7

In task 7 I create the functions “estimate\_and\_plot()” and “plot\_estimated\_vs\_true\_loads()”/

estimate\_and\_plot() starts by separating the heating targets from the cooling targets, using all data. A model is fit for each target using the previously calculated optimal degrees, and then tested against predictions.

From here it calls plot\_estimated\_vs\_true\_loads to plot the data.

Finally I calculate and output the mean absolute error using the method from \_regression.

A screen shot of a computer code

Description automatically generated

As we can see, cooling predictions are more accurate than heating, with a lower absolute difference.

A screen shot of a computer

Description automatically generated

To plot the data, I scatter Heating data in Orange, and Cooling data in Blue. I then plot a reference line, from the min to max of each target.

A screen shot of a computer program

Description automatically generated

We can our observations are consistent with the visualization. In most cases cooling trends closer to the reference line than Heating.

A graph showing a comparison of a load

Description automatically generated with medium confidence