cw2 – brook raindle 19039840

## Pre-Processing

Preprocessing is extremely important, it is the step of transforming raw data into a format that is suitable for analysis and modelling. It includes handling missing values, feature selection, feature scaling, and data transformation

Figure 1 – Example code used for pre-processing steps.

A computer code on a black background

Description automatically generated

Missing data is an issue and can affect the performance of my machine-learning algorithm. Missing values are addressed using the fillna method with the 'ffill' parameter, which propagates the last valid observation forward along the column axis. This approach maintains temporal continuity while filling in missing values, with a limit of 5 consecutive missing values filled. By handling missing values in this manner, the integrity and completeness of the dataset are preserved.

In addition to handling missing values, selecting relevant features, and scaling the data, the code includes a data transformation step in the pre\_pro\_heated function. Here, the indoor temperature is increased by 0.5 degrees, possibly to simulate a heated indoor environment for specific analyses or scenarios. This transformation highlights the flexibility of pre-processing techniques in accommodating domain-specific requirements and enhancing the interpretability of the data.

Feature selection is essential for focusing the analysis on relevant variables that have a significant impact on the target variable. The code demonstrates feature selection by specifying columns of interest for further processing. For example, in the pre\_pro\_window function, columns related to ‘outside temperature’ and ‘outdoor relative humidity sensor’ readings are selected. Similarly, in other functions such as ‘pre\_pro\_close\_win’ and ‘pre\_pro\_heated’, columns pertaining to ‘indoor temperature’ and ‘humidity’ are chosen. This targeted approach to feature selection helps in extracting meaningful information from the dataset while reducing noise and dimensionality, the decision-making that led me to this was that during data analysis in CW1, it could be seen that only the temperature has much of an effect on the satisfaction, however, humidity also shows feint signs of a relationship, so I also included that.

Feature scaling standardizes the range of features in the dataset. Standardization ensures that all features contribute equally to the analysis and prevents features with larger scales from dominating the model training process. I use the StandardScaler from the sklearn.preprocessing module to standardize the selected features. By scaling the features to have a mean of 0 and a standard deviation of 1, the data is brought onto a common scale, facilitating more effective model training and prediction. A new dataframe is built using the pandas library, and each row of data is collected using the requested features, named ‘columns\_to\_include’.

The Satisfaction, however, is not scaled. Instead, it is stored as a label variable to train the data against.

## Training the models

**Splitting Data:** The pre-processed features are split into training and validation sets using the train\_test\_split function from scikit-learn. This function divides the data into random train and validation subsets, with 80% of the data used for training and 20% for validation.

**Model Architecture:** A feed-forward neural network model is defined using Keras' Sequential API. It consists of three layers: two dense layers with ReLU activation functions and one output layer.

**Compiler**: The model is compiled using the Adam optimizer and mean squared error loss function.

**Training**: The model is trained using the fit method, with the training inputs, training labels, validation inputs, and validation labels. The training process continues for 1000 epochs with a batch size of 32.

A screen shot of a computer program

Description automatically generated**Evaluation**: The trained model's performance is evaluated on the validation set using the evaluate method, and the mean squared error is printed.

Figure 2 – split\_fit\_data() method, a template for each model to use for training.

**Training System for Closed Windows:**

The training process for the system predicting satisfaction when windows are closed follows the same steps as outlined above for open windows, with the only difference being the data pre-processing function used (pre\_pro\_close\_win).

**Training System for Heated Environment:**

The training process for the system predicting satisfaction in a heated environment also follows the same steps as outlined above for open windows, with the only difference being the data pre-processing function used (pre\_pro\_heated).

**Saving Models:**

After training each system, the trained models are saved to disk using the save method provided by Keras. This allows for easy retrieval and reuse of the trained models for inference or further use.

## Optimizing Machines for New Day Data

To control the machines, and know when the time to activate the heater, requires future data, in this scenario, we have access to ‘New\_day.csv’ which contains a set of test data for each 15-minute interval throughout the day. Firstly, we pre-process this data, removing any missing data, and feeding it through all 3 pre-processing strategies, so we can run the satisfaction prediction models against the test data and produce a set of comparable numbers, reflecting the satisfaction that an occupant will display, under the given conditions, including Open Window (or Ventilation), Activated heating element, And Resting temperatures (Close Window). Using the same ‘pre\_pro\_close\_win’ etc.… methods displayed earlier, to perform the preprocessing.

Figure 3 - Code used to predict and assign each satisfaction model after preprocessing

A screen shot of a computer code

Description automatically generated

The models are predicted after checking the test data has been completed, by seeing if each model's features have been retrieved from the test data. it then prints to the console for viewing. I also chose to save the entire prediction to a copy of the original test dataset. So a CSV file containing the test dataset, with classified Satisfaction values for Open window, Closed Window, and Heated, is provided and can be used to compare and make further decisions. (‘New\_day\_predicted.csv’)

Figure 4 – Logic for Deciding when to activate the heater

A screen shot of a computer code

Description automatically generated

To summarize the above, it calculates how many different 4-hour windows could occur, given a 15-minute timestep. It then calculates the average satisfaction over those 4 hours (Closed windows) and compares it to the average satisfaction of the same time window but from the heated satisfaction prediction. It will then add it to the List of average satisfactions, a collection of each 4-hour timewindows, increases. After all have been calculated, it is sorted in ascending order.

The top timeslot is automatically assigned as one of the chosen 2, as it doesn’t have any conflicting times, then the rest of the average satisfaction increases are fed through a process of elimination until one that does not conflict with the already chosen timeslot. This is then assigned as the second timeslot

Lastly, we have to discover when to open the window (Ventilation). This follows a slightly more complex logic, as we have to account for several more things. I decided to iterate over the data in 30-minute intervals since opening a window has complicated rules. For the first 15 minutes, the temperature drops to external temperatures. Once closing the window, it takes another 15 minutes until the temperature returns to the resting values. Also, I don’t want to overlap opening the window with turning the heating on, so an if statement is included to check against the already existing timeslots of heating. This is the reason why the heating is calculated first because it is a finite resource which should be optimized where possible, whereas the window can be opened at no cost.

If the Average satisfaction increase is positive for the 30-minute time-slot when compared to the closed window satisfaction average. Then the system will add the timeslot to the array of best opening closing windows list.   
  
I decided to only perform a text-based simulation as I was able to aptly display all the useful info that instructs a system as to whether the heating is active or whether the window is open, as seen in figure 5

Figure 5 – example output of ran simulation

A screen shot of a computer

Description automatically generated

As seen above, you can see when each element is active, and their respective satisfaction outputs based on the state of the environment. The console displays every 15 minute timeslot. It also prints the chosen timeslots pre-emptively as seen in figure 6, if you do not wish to search through the 15 minute slots.

A screen shot of a computer screen

Description automatically generated

Figure 6 – Alternate output text

A graph of different colored lines

Description automatically generated

Figure 7 – Plotted satisfactions from predicted test data csv.

Figure 7 Is Extra information to view the Satisfaction compared to each projected satisfaction level. It can be seen that at any time, only 3 colours are visible, which means the optimizes satisfaction overlaps, meaning it choses when each room state is optimal. Each of the large peaks are used, rarely dipping into the lower satisfaction levels that occur when the windows are closed, however sometimes it is unable to use the heated satisfaction as it has a cost to using and mayube that time slot didn’t benefit the overall satisfaction for the day as much as potential other opportunies can provide.