**EMATM0044 Introduction to AI Coursework Part 1**

**Introduction**

The problem I have is to predict the net hourly energy of the plant (PE), where I am given a csv file with 4 other variables (columns) of data that can be used to calculate the PE column. All columns in my csv file contain continuous data and the output column that I am predicting is also continuous. This tells me this is a regression problem as I am modelling a dependent variable (Y), whilst using variables (X) that are trying to predict the output variable (Y). Therefore, I will use regression models that use the X variables to predict the Y variable.

**Methods**

I will be using and comparing 3 different algorithms that are all separate and provide different ways of predicting output variables in regression problems. The first algorithm I will use is linear regression. This algorithm is a basic one but is always a good starting algorithm to use when tackling simple regression problems and can also be used as a baseline to compare to when using more complex algorithms. The linear regression algorithm can be very useful when there are linear relationships in the X variables and Y variable. In order to look specifically at these relationships, I created a correlation matrix heatmap that shows which variables are correlated with each other, if at all.

A picture containing chart

Description automatically generated

As shown by the correlation heatmap, no variables have perfect linear correlations with each other how some are close to linear, such as AT and V, along with V and PE almost being perfectly negatively correlated to each other. Overall, this heatmap shows that many variables seem to be in the range of |0, 0.5| correlation coefficient showing that linear regression may not perform the best for this regression problem, however it is still worth testing.

The next algorithm I will use is polynomial regression which is a form of linear regression but instead of assuming linearity, polynomial regression uses a hyperparameter called nth degree which transforms the function into a different polynomial function (e.g 3 degrees is a cubic model). This algorithm I assume to perform slightly better than the linear regression due to its ability to adjust to non-linearity.

The final algorithm I will use is random forest. This a much more complex algorithm that uses multiple hyperparameters that ultimately decide how the construction of the decision trees is formed. I expect the random forest model to perform the best out of the 3 due to its complex nature and ability to provide high accuracy for all multitudes of regression problems.

I decided to use two performance metrics for my analysis because I wanted to compare the results and using only one performance metric can be consequential if one of the performance metrics does not work well with a certain model, for example being sensitive to outliers. The first metric I decided to use was R – squared (R2). I chose this performance metric due to its ability to give me an idea of how the model has fit the dataset. It gives me a value in the range of (1, -1) meaning that if I got a value close to 1, I would know that the model has fit the dataset well. R2 is also very useful for all sizes to datasets and is very easy to interpret. The second performance metric I decided to compare it with is Mean Squared Error (MSE). This metric is slightly different to R2 in that it gives me a value for squared difference between the actual and predicted value. MSE is sensitive to outliers, however with this dataset I am not expecting many so it should not affect the results much, especially with cross-validation.

Baseline model – sklearn dummy model

I chose sklearns dummy regression model to compare my models against as a baseline. The dummy model from sklearn is a very simple model and should almost never outperform real regression problems. The main reason for using this dummy regression model is to see if the regression models I chose are outperforming it. Especially for the more complex models such as polynomial regression and random forest regression that involve hyperparameters, they may be able to capture relationships between the X variables and target Y variable that the dummy model cannot. If I can see that my chosen models are significantly outperforming the dummy variable then they are fine to use.

Two of the models I have chosen use hyperparameters, polynomial regression only uses 1 which is ‘degree’ and random forest regression uses numerous. Hyperparameters are parameters that models use that can significantly alter the behaviour of the underlying algorithm. They are set before training the machine learning model which is what separates them from parameters that are learnt during training. Finding the optimal hyperparameters for a model is very important as it can drastically change the performance of the model on the dataset and improve the model’s predictability. The way I chose my hyperparameters for both models was through cross-validation and grid search. Polynomial regression was significantly easier to tune due to it only having a singular hyperparameter ‘degree’. I knew that when I start increasing the degree hyperparameter there is more chance of my model overfitting the training data, causing poor performance for the unseen data. I decided to first cross-validate the training data using KFold cross-validation with 5 folds for more accurate results. After cross-validating I performed a grid search where I set the parameter grid to include degrees in the range of (1,10). After fitting the grid search on the training data I found that the best degree was 4.

Chart, line chart

Description automatically generated

This graph shows how the value for MSE changes with degree. The optimal degree is 4 and you can see that after 8 MSE is started to decrease drastically showing me that if I performed the grid search with more than 10 degrees the model would start to overfit the data which is what I planned on avoiding.

For random forest regression I decided to follow a very similar method for hyperparameter tuning. Random Forest regression is a more complex model that uses several hyperparameters so it will be significantly difficult to find the optimal hyperparameters without overfitting. It was important to initially decide what are the most important hyperparameters to tune for that will have the biggest impact on the model’s performance. A grid search can take a very long time when tuning for several hyperparameters at once so I needed to decide on a set that would be the most beneficial.

Here is the list I decided on:

‘n\_estimators’ – number of trees in the forest.

‘Max\_depth’ – depth of each tree – can lead to overfitting.

‘Max\_features’ – number of features to look for.

‘Min\_samples\_leaf’ – min. number of samples to be at a leaf node – greater value can help reduce overfitting.

‘Min\_samples\_split’ – min. number of samples to split a node – greater value can help reduce overfitting due to simplicity of trees.

‘Bootstrap’ – ‘True’ allows whole dataset to build each tree.

Chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated

Chart

Description automatically generated

Chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated

The plots above show how each value for the tuned hyperparameters performed throughout the gridsearch. The y-axis is the performance metric which I chose as MSE for these plots and the x-axis is the parameter combination index which is the individual point they are at during the gridsearch. These plots gave me an insight to how important each value in the hyperparameter is. For example, I can see that max\_depth = 5 is significantly worse than the other values, however for other hyperparameters such as min\_samples\_leaf and min\_samples\_split the difference is marginal for each value. After testing several values for hyperparameters using gridsearchCV, I found optimal values for each hyperparameter I tuned that gave me the best values for my performance metrics MSE and R2.

At the start of my analysis, I split my data into three different datasets:

Training set: 64% - used to train my models with

Validation set: 16% - used to test the performance of my models throughout the analysis and make predictions on.

Test set: 20% - saved for my final predictions and valuation after I have trained all models.