University of BRISTOL

Thomas Rowland - 1837499 - EMATM0044_2022

I. INTRODUCTION

Inspection of the dataset shows 5 columns of data representing the features of a combined cycle power plant over 6 years. The 5 features are AT (Average Temperature), V (Exhaust Vacuum), AP (Ambient Pressure), RH (Relative Humidity) and PE (net hourly electrical output). The total dataset 9569 rows of data which can be split into appropriate training and testing datasets.

This task is a regression problem. The goal is to use the first 4 features (AT, V, AP, RH) to make a prediction of PE, the electrical output. This is a regression problem instead of a classification or clustering problem because the target value, PE, is a continuous quantity. The objective of the machine learning system is not to categorise the observations into classes but instead to predict a numerical value for the electrical output of each hour based on the values of the other 4 features.

II. METHOD

A. Dataset Pre-processing

The dataset was first visually inspected to determine any immediate problems and to establish the corresc train-test split. It was shown that the was a total of 9569 items in the dataset, which would be split down as 70% training, 21% validation, 9% test. The validation dataset will be used throughout the training processes but the test set will remain unseen for the final evaluation. The datasets where also scaled to improve stability.

B. Algorithms

The algorithms chosen for comparison where a Neural Network based algorithm and K-Nearest Neighbours. These algorithms will be compared against each other and against a Dummy Model.

1. Dummy Model

A dummy model (or baseline model) is a very simple model that is used to compare with your actual model. It is used to set a baseline performance which your model will look to improve up on.

As this is a regression problem, Scikit-learn has a 'DummyRegressor'[1]. This model will make predictions based on simple parameters which in this case are *mean*, *median*, *quantile* and *constant*. For this purpose the *mean* was chosen, so the model will predict the mean of the dataset.

When comparing algorithms to the baseline algorithm, if the tested algorithm performs worse it could be an easy indication that it has been implemented wrong or that it is not suitable for the task.

2. Neural Network

Neural networks are constructed by connecting layers of "neurons" between an input and an output. Each neuron takes an input applies a weight multiplier to it, adds a bias term and then passes it through an activation function which then serves as the input to the next layer in the network. This continues until the activations have reached the final 'output' layer whereby the activation function forms the results into a prediction.

For our purpose, the inputs for the network are the values for AT, V, AP and RH and the output is the predicted value for PE. The weights and biases in each neuron or unit are continuously updated throughout the training process where the network is repeatedly presented with examples from the training set and validation set. It then adjusts its parameters to minimise the difference between its predictions and the ground truth values.

Neural networks are good for learning complex relationships in data. Their complexity can make them difficult to interpret and are often troubled by problems caused by limited training data which can cause overfitting. Additionally there are many hyperparameters that should be tuned to optimise performance such as the number of layers, the number of units in each layer, the optimiser, training epochs and the learning rate of that optimiser.

3. K-Nearest Neighbours

The neural network was an iteration-based learning algorithm, conversely K-Nearest Neighbours (KNN) is a instance based learning algorithm. It works by storing all instances of the training data. When a new data point is seen the

algorithm looks at the 'K' closest data points - the nearest neighbours - and either assigns the most common or the the average outcome depending on whether the problem is a classification or regression problem.

For this regression problem, the KNN algorithm would look at the 'K' training examples that have the most similar values for AT, V, AP and RH for the given test values. It then calculates the predicted value of PE by averaging the values of PE from these the 'K' training examples.

Unlike neural networks the KNN algorithm is relatively simple and easy to understand and implement. Although it requires no training it is still computationally expensive during testing and inference as it computes the distances between the test values and every single training example.

C. Hyperparameter Optimisation

The optimal hyperparameters will be identified using a combination of grid search, random search techniques and manual tuning. The performance of the models with the optimised hyperparameters will be evaluated using the RMSE and R^2 metrics.

Grid Search: This method exhausts all combination of a pre-defined set of hyperparameters and evaluates the model performance for each combination. As it looks at every combination it is very computationally expensive and can only be used with a small hyperparameter space, it will however give the optimal solution.

Random Search: If the hyperparameter space is large a strategy such as Random search could be employed. It uses random combinations from a predefined workspace and evaluates the model performance. This method is typically faster but there is not guarantee of the optimal solution.

Manual Tuning: This method involves manually changing the hyperparameters based on the models performance. It requires some knowledge about what might work well given the nature of the problem. It is slow and requires the user to manually track the difference a change makes.

1. Neural Network Hyperparameters

For the Neural Network, the hyperparameters considered in this study are:

Number of Layers: The depth of the neural network. The number of layers will have an impact on underfitting and overfitting.

Number of Neurons: How many neurons are in each layer. This should be changed depending on the complexity of the data.

Learning Rate: How quickly the algorithm converges the weights to the optimal values. If this is too small it could get stuck in a false minimum and too large could result in an overshoot.

Activation Function: This determines the output of a neuron given a set of inputs.

Optimiser: The optimiser is responsible for updating the weights in the model given the error.

Initial testing was carried out with the Keras Tuner [2], a framework within the keras API that allows the hyperparameter space to be defined and then iterated over with strategies such as random search or bayesian optimisation. Table I illustrates the search space. A random search strategy was employed in this study.

Hyperparameter	Description	Search Space
num_layers	Number of Dense layers in the model	1 to 4
units_i	Number of neurons in i-th Dense layer	4 to 32 (step 4) for each layer
learning_rate	Learning rate of the RMSprop optimizer	1e-2, 5e-4, 1e-3, 1e-4, 1e-5

TABLE I. Hyperparameters in the Keras Tuner Search Space for the Neural Network

Following the automated approach with the keras tuner, a manual tuning approach was employed to optimise the model. Table II shows the iterations attempted to improve the system. For every system with both tuning methods the batch size was 16 and the model was trained for 50 epochs.

2. K-Nearest Neighbors Hyperparameters

For the K-Nearest Neighbors, the hyperparameters considered are:

Number of Neighbors (K): An important hyperparameter for KNN. Smaller values of K make it more sensitive to noise whereas larger values make it more computationally expensive and potentially less accurate by including more dissimilar examples.

Distance Metric: Different distance metrics can be used in the KNN algorithm. Options include Euclidean distance, Manhattan distance, and Minkowski distance.

Model	Number of Layers	Neurons per Layer	Activation Function	Regularizer	Loss Function
keras_DNN	5	24, 8, 8, 32, 1	relu, relu, relu, relu, -	-	mean_squared_error
DNN	3	64, 34, 1	LeakyReLU, LeakyReLU, -	l2(0.01)	mean_squared_error
DNN2	2	64, 1	LeakyReLU, -	l2(0.01)	mean_squared_error
DNN3	2	16, 1	LeakyReLU, -	l2(0.01)	mean_squared_error
DNN4	2	16, 1	LeakyReLU, -	l2(0.01)	mean_squared_error
DNN5	4	16, 8, 1	LeakyReLU, LeakyReLU, -	l2(0.01)	mean_squared_error
DNN6	2	16, 1	LeakyReLU, -	l2(0.01)	huber_loss
DNN7	2	16, 1	LeakyReLU, -	l2(0.01)	mean_absolute_error

TABLE II. Manual Hyperparameter Tuning for DNN Models

Weight Function: This determines how the model values the weight contributions of the neighbours. There generally is the choice of uniform (equally weighted) or distance-based (nearer points have greater influence).

Optimising the more limited KNN optimiser is simpler than with the neural network because it does not require lengthy training. A grid search algorithm was employed to test every combination within the search space. The KNN search space is shown in Table III.

Hyperparameter	Description	Search Space
n_neighbors	Number of neighbors to use	1 to 24
weights	Weight function used in prediction	uniform, distance
metric	The distance metric to use	euclidean, manhattan, minkowski

TABLE III. Hyperparameters in the Grid Search for KNeighborsRegressor

D. Performance Metrics

Performance metrics are used by models to assess the quality of a predictions. For regression tasks there are several metrics that are suitable such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RSME) and R^2. For this study the chosen metrics were MSE and R^2 [3].

- *Root Mean Squared Error (RMSE):* RMSE emphasises larger errors by squaring the difference and averaging them. Lower RSME values represent better performance with 0 being the ideas score and a perfect prediction. The result is also more understandable as the error is in the same units as the target variable which this case will be KWh.
- *Coefficient of Determination (R^2):* R^2 is a statistical measure indicating the proportion of the dependant variables variance. The value ranges from negatve values to a maximum of 1, with 1 being the best possible score. A negative score indicates a very poorly performing model.

These metrics will be used to assess the models performance. The RSME will show the average magnitude of the predicted errors, giving a sense of how far the predictions are off on average. The R^2 score can show how well the variation in PE can be explained by the model. Both of these combined should give a good view of the model performance.

III. RESULTS AND ANALYSIS

Results from the the neural networks were generally promising. They showed that they had succeeded in the learning information from the features and could make a reasonably accurate prediction based on the input data. Table IV shows the results from each iteration. The results show that the optimisation improved the training and validation loss but didn't significantly reduce the R^2 or the RSME. The iterations did however seem to affect how long the model took to converge to a solution. The graphs and tables do not appear to show any overfitting as both the training loss and validation loss are similar in magnitude. With more iterations a more optimal solution could have been found but it is unlikely to improve by much as the model is already accurate.

Optimisation of the KNN algorithm was significantly quicker. The KNN optimisation involved cross-validation - dividing the training set into 5. Four of these get used as a training set and 1 for validation, this then gets repeated 5 times. Cross-validation makes the estimate more reliable by reducing the variance of a single trial [4].

Model	Final Training Loss	Final Validation Loss	RMSE	R-squared
Keras_DNN	20.525	18.778	4.333	0.936
DNN1	24.193	22.057	4.245	0.939
DNN2	24.558	24.611	4.260	0.939
DNN3	27.076	26.706	4.332	0.936
DNN4	26.512	25.544	4.305	0.937
DNN5	22.693	25.233	4.804	0.922
DNN6	8.298	8.246	4.379	0.935
DNN7	8.590	8.550	4.432	0.934

TABLE IV. Model Performance Comparison

The best hyperparameters found were k = 8, weights = 'uniform', and metric = 'manhattan'. These parameters provided the lowest RMSE and an R^2 value close to 1, showing that the algorithm was working well on the data.

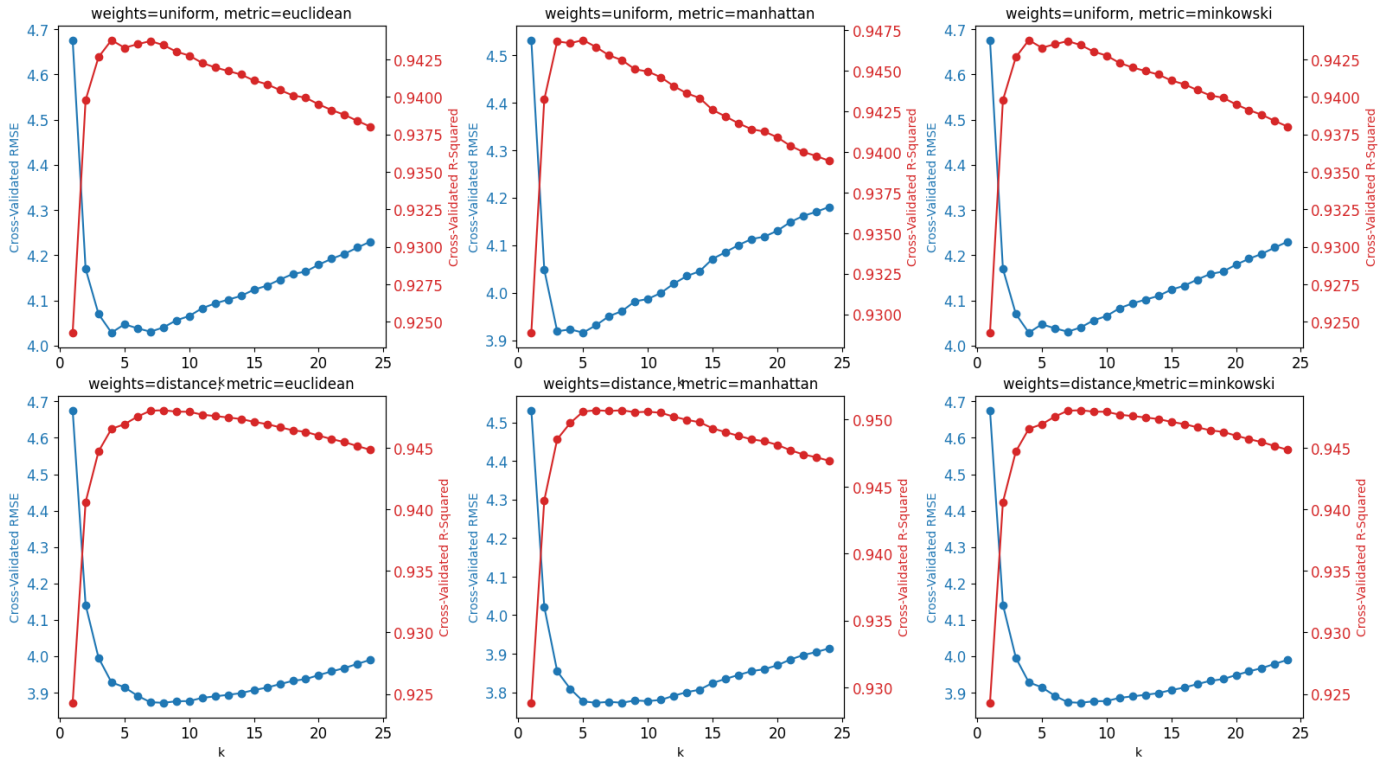

FIG. 1. The results from the grid search hyperparameter optimisation on the KNN algorithm. See Table III for the search space.

Model	R-Squared	RMSE
Dummy Model	-0.000204	17.26
Best DNN	0.9425	4.022
KNN	0.9639	3.280

TABLE V. Comparison of R-Squared and RMSE for Different Models

After optimisation all three models were tested against a completely unseen test dataset. This dataset consisted of 862 rows of data. The results in Table V show the how each model performed. It can be seen that both models significantly outperformed the Dummy Model with the KNN performing the best. With a R^2 score of near 1 and a fairly low RMSE the models appear to be able to make accurate predictions on the data.

For further improvements more data should be collected. Neural networks are powerful and adaptable tools but they require large amounts of data. More data would give greater opportunities to learn the underlying patterns which could improve the performance over the KNN.

REFERENCES

[1] scikit-learn developers, *Model evaluation: Quantifying the quality of predictions*, https://scikit-learn.org/stable/modules/model_evaluation.html#dummy-estimators, Accessed: 2023-05-24, 2023.

[2] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi, *et al.*, *Kerastuner*, https://github.com/keras-team/keras-tuner, 2019.

[3] M. M. Rahman, D. Berger, and J. Levman, "Novel metrics for evaluation and validation of regression-based supervised learning," in *2022 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*, 2022, pp. 1–6. DOI: 10.1109/CSDE56538.2022.10089291.

[4] I. Tembo, *Cross-validation using k-fold with scikit-learn*, Mar. 2022. [Online]. Available: https://isheunesu48.medium.com/cross-validation-using-k-fold-with-scikit-learn-cfc44bf1ce6.