Assignment 2 – Regression using Scikit-learn

Student Name: <Hameed Adagun> **Student ID**: <20329901> **Programme**: <4BCT>

Algorithm 1 - <k Nearest Neighbours (kNN)>

<Introduction> **kNN** is an algorithm that is given a query case, and a value is predicted. It calculates the distance between the query case and other stored training cases. The number of stored training cases used depends on the **k** (the number of nearest neighbors). This is used to calculate the value predicted. This is done on all the test cases. It can be used for **classification and regression tasks**. For regression tasks the value predicted would be a **numerical value** while for classification tasks the value predicted would be a **categorical value**.

Detailed Description of Algorithm 1. (Uses References [1], [2] and [3])

Using a **low value of k** makes the algorithm **susceptible to noise** causing it too **overfit**. This means that the algorithm fits too well with the training data and will not perform well on unseen data. While using a **high value of k** will have **a smoothing effect** making the algorithm to **underfit**. This may cause the algorithm to miss out on patterns in the dataset making it too simplistic.

The **weight** for the hyperparameter you use decides how much the distance between each neighbour impact the result for the query case. There are two different weights you can use for the hyperparameter: **uniform weighting** and **distance weighting**. For the **uniform weighting** each neighbour has equal weighing meaning that the distance **does not matter**. For the **distance weighting** each neighbour is given a weigh based on the inverse square of its distance from the query. In this instance the **distance does matter**. We use the inverse square so that we don't get any negative distances. ***See Appendix [1]*** to see the formula for both weights.

There are many ways to calculate the distance between the query case and each of the number of neighbors. This is hyperparameter, under skicit learn this is called **'metric'**. The three main metrics that are used for this hyperparameter are **Minkowski, Euclidean** and **Manhattan**. The default used for the hyperparameter is Minkowski. ***See Appendix [2]*** for the formula for each of these metrics.

Figure 1: Graphic illustration of the algorithm works

Hyperparameter Details for Tuning.

Number of Neighbours: k is the number of neighbours that is used. The default for this in skicit learn is 5. It is key that we use the right number for k otherwise the algorithm can easily overfit or underfit.

Weights: The weight for the hyperparameter you use decides how much the distance between each neighbour impact the result for the query case.

Algorithm 2 - <Neural Networks>

<Introduction> Neural networks have layers of neurons. **Neurons** are like nodes that are based on weights, biases and activation functions. These layers are organised into the input, hidden and output layer. The neurons are interconnected to each other in different layers. During training, the network adjusts the **weights and biases**. The network learns from its mistakes by minimizing the difference between its predictions and the actual outcomes.

A **weight** is a parameter that adjusts the **strength of the connection between two neurons**. When you have a neural network layer, each connection between neurons has a weight associated with it. During training, these **weights are adjusted** to optimize the model's performance in making predictions.

A **bias** is an additional parameter in a neuron that allows for some flexibility and helps the model account for situations where all **inputs might be zero**. In this scenario the input values don't provide sufficient information on their own. Including biases in a neural network allows it to **learn more complex patterns** and relationships in the data.

Once we set up the input layer, we give weights to different variables to see how much they matter. Bigger weights mean something is more important for the output. We multiply all the inputs by their weights, add them up, and then use an activation function to decide the output. If the output is higher than a specified threshold, the **neuron gets activated** and passes information to the next layer.

Detailed Description of Algorithm 2. (Uses References [4], [9], [10], [11] and [12])

Activation functions decides what functions is used for the hidden layer. They impact the neural network in many ways. One way is by introducing **non-linearity** into the network. If we used only linear activation functions, no matter how many layers we add, the entire network would behave like a single-layer network. Non-linearity allows us to **learn more complex relationships** in data. The output range for activation functions also differ making some more suitable for different tasks.

Hidden layer shape represents the number of neurons in the hidden layer. A **large hidden layer shape** makes the neural network **more complex**. However, having too many neurons make cause the network to **overfit**, making it too tailored to the training data and performing bad on new unseen. Conversely, having **not enough neurons** may make the network too **simplistic** causing it to **underfit**.

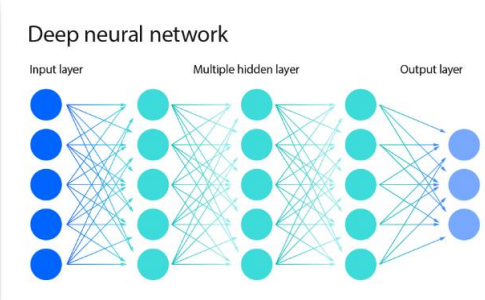

Figure 2: Graphic illustration [11] of the algorithm

Hyperparameter Details for Tuning.

Hidden layer shape: The number of neurons in the hidden layer. The default is for this is (100,)

Activation function: The function used for the hidden layer. The default function is 'relu'. The other functions include 'identify', 'logistic' and 'tanh'.

Algorithm 1 - <k Nearest Neighbours (kNN)> - Model Training and Evaluation

<Introduction> Unlike the classification assignment where we randomly split the dataset into the training and testing set, in this assignment I am using **10-fold cross validation**. This is to help with potentially having all the difficulty examples in the training set and the easy example in the testing set. For my dataset how this would work is by having **10 different folds**. Each fold represents the testing data, and they never overlap. The fold would be 1/10 of the dataset and the rest would be the training data. I would get the score for the testing and training data and then repeat the process 10 times using different folds for the testing data.

I needed to choose one domain specific and domain independent measure of error to evaluate my model. The one **domain independent measure of error** I used was **R-squared**, it is a statistical measure that represents the proportion of the variance in the dependent variable that is predictable from the independent variables. The **domain specific measure of error** I used was **MSE (Mean Squared Error)** which measures the average squared error between the predicted and actual values. It is squared so there aren't any negative numbers for the error. The lower the error the more accurate the model as there is less of a difference between the predicted and actual value.

Visualisation

Created a subset using two features to visualise the dataset. I used the default hyperparameters for the kNN model which is the number of neighbor being 5 and the weight being uniform. The two features I used was **'percent_nickel'** and **'percent_copper'**. For the training dataset the **r2 score** I got was **0.46** and

the **mse score** I got was **4489.52**. For the testing dataset the **r2 score** I got was **-0.49** and the **mse score** I got was **9328.33.** The result for this model is very poor, this may be because the dataset used to train the model was small as it only used two features, not allowing the model to learn more patterns.

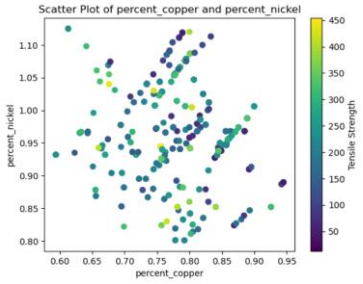

Figure 3: Visualisation of the training dataset before hyperparameter tuning

Training and Evaluation Details

The hyperparameters I tuned for the kNN model was the number of neighbours and weights. To do this I used a nested for loop. In the outer loop I looped through the weights and in the inner loop I looped through the number of neighbors. The value for the number of neighbor was 1-10 and the values for the weights were uniform and distance. I could have used grid search cv to do this but decided to use nested loops.

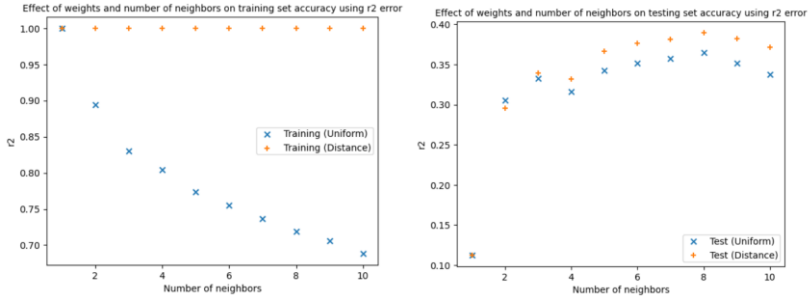

Figure 4: Summary of Results Achieved from Training and Testing (using r2)

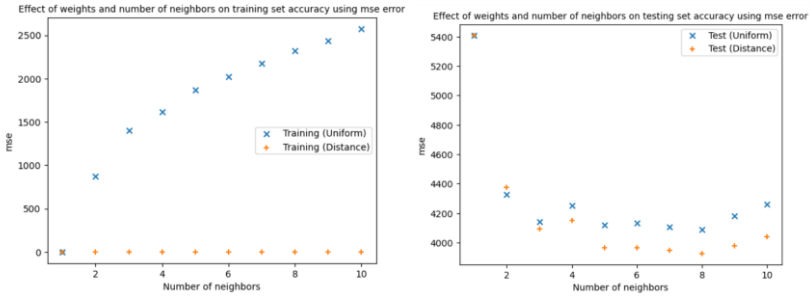

Figure 5: Summary of Results Achieved from Training and Testing (using mse)

Discussion of results

Based on the training on the evaluation details the best hyperparameters for the training set would be the weights as distance and the number of neighbours ranging from 1-10. For the r2 error the result for these hyperparameters was 1.0. This is the maximum value which shows very good performance. For the mse the result was 0. This shows that there no squared error between the predicted and actual values. Which shows that the model is perfect on the training dataset.

For the testing set the best hyperparameters would be distance for the weight and 8 for the number of neighbours. For the r2 error the result for these hyperparameters was 0.3897. This was higher than the all the other hyperparameters for this measure of error, which indicates better performance. For the mse the result was 3926.13. This was lower than the rest of the other hyperparameters. This shows that it is the best performing as less of a squared difference between the predicted and actual value.

Algorithm 2 - <Neural Networks> - Model Training and Evaluation

<Introduction> Similar to kNN.

Visualisation

To visualise the neural is used two features similarly to the KNN model. The two features I used was **'percent_nickel'** and **'percent_copper',** the same as the kNN model. To do this I used a dependency called networkx. I created an input, hidden and output layer. For this model I used the default hyperparameters. The activation function was 'relu' and the hidden layer shape was (100,). For the training dataset the **r2 score** I got was **-1.38** and the **mse score** I got was **19670.24**. For the testing dataset the **r2 score** I got was **-1.85** and the **mse score** I got was 1**9793.44.** These poor results may be caused by the small size of the dataset like the kNN model that used two features.

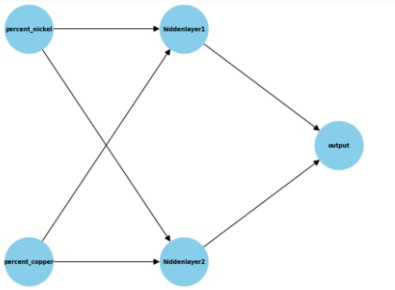

Figure 5: Visualisation of the dataset

Training and Evaluation Details

Similarly, to the KNN model I used nested loops to iterate through the two hyperparameters I was tuning. The two hyperparameters I was tuning were the activation functions and the shape of the hidden layers. The values for the activation functions were 'relu', 'identity', 'logistic' and 'tanh'. The values used for the hidden layer shape were (96-105,).

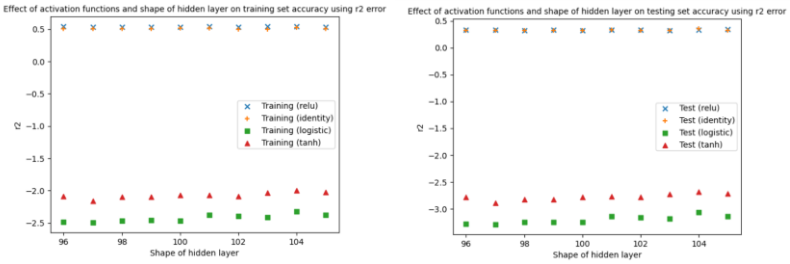

Figure 6: Summary of Results Achieved from Training and Testing (using r2)

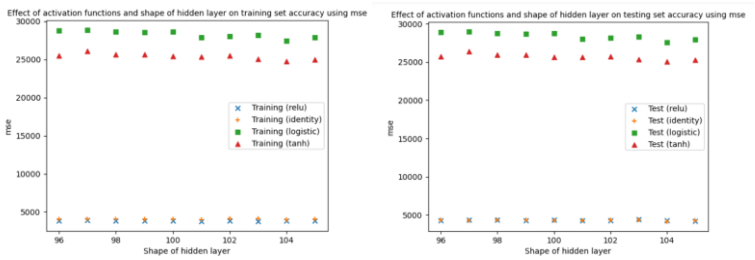

Figure 7: Summary of Results Achieved from Training and Testing (using r2)

Discussion of results

For the training set the best hyperparameters would be the activation function as relu and the shape of the hidden layer as 101. For the r2 error the result for these hyperparameters was **0.543**. This is the highest value compared to the other hyperparameter. For the mse the result was **3771.89**. This was lowest value which show it was the best performing.

For the testing set the best hyperparameters would be activation function as identity and the shape of the hidden layer as 104. For the r2 error the result for these hyperparameters was **0.354**. This was higher than the all the other hyperparameters for this measure of error, which indicates better performance. For the mse the result was **4155.85**. This was lower than the rest of the other hyperparameters. This shows that it is the best performing as less of a squared difference between the predicted and actual value.

Conclusions

Based on the results I believe the best hyperparameters for the KNN model is using **distance** as the weight and having the **number of neighbours as 8**. Using these hyperparameters you get the best results for the training dataset as well as the test dataset. It is interesting to note that the number of neighbours is a lot more sensitive when uniform as a weight. This can be seen as the measure of errors start drastically when changing the number of neighbours. Also, when you use uniform as a weight, and you increase the number of neighbours the models start to perform poorly. This is because the model is underfitting making the model miss out on patterns.

For the neural network I believe the best hyperparameters is using **relu** as the activation function and the shape of the hidden as **104**. These may not be the best hyperparameters for the testing dataset, but I believe the difference in performance between best hyperparameters for the test dataset and the performance with these hyperparameters are so minimal that it does not really matter. The activation

function hyperparameters is a lot more sensitive than the shape of the hidden layer. The difference in performance when using either relu or identity compared to using logistic or tanh is a lot more than when you change the shape of the of the hidden layer. I believe the model underfits when you use logistic or tanh as an activation function when causes the performance for the training and testing dataset.

Comparative Analysis of Algorithm Performances

As I used the same two measure of errors for the two algorithms, I am able to directly compare the two of them. Looking at the results it is clear to see that the kNN model is much better than the neural network model for this dataset.

```
Average Train r2 Score with weights distance and 8 neighbors: 1.0
Average Train mean squared error Score with weights distance and 8 neighbors: 0.0
Average Test r2 Score with weights distance and 8 neighbors: 0.38973937344688675
Average Test mean squared error Score with weights distance and 8 neighbors: 3926.130992121347
```

```
Average Train r2 Score with activation function relu and 104 shape: 0.5392860790981137
Average Train mean_squared_error Score with activation function relu and 104 shape: 3804.7586629750745
Average Test r2 Score with activation function relu and 104 shape: 0.33199228007769027
Average Test mean_squared_error Score with activation function relu and 104 shape: 4295.354922775272
```

Both the r2 score and the mse perform better for the training and testing dataset when using the kNN model.