ENPM808A - Introduction to Machine Learning

Final Project

SURIYA SURESH

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Chapter 1

INTRODUCTION

A training and testing data set consisting of Lasar range data from a car-like robot model were given. It consisted of a 1D array of 1080 values, each value representing a degree value of 0.25 degrees, totaling 270 degrees field of view of the Laser data. An additional 12 columns represent the cartesian coordinates for position and a quaternion ($q_r$andq_k).These make up the input x data. The final two rows are the linear and angular velocities, making for a total of 1094 columns of the training and test datasets.

The aim of the project is to predict the linear and angular velocity of the car using a machine-learning algorithm.

1.1 Models Considered

The models that have been considered for this report are

- Linear Regression
- Decision Tree Regression
- Neural Networks

Linear regression and Decision Tree Regression were taken from the python package Sklearn and Neural Networks from Keras.

Chapter 2

DATA PREPROCESSING

A subset of the provided training dataset sizing about 5GB was used as the given training dataset was 19GB and required a lot of resources to be processed. The testing dataset was used as such in its entirety. For checking the presence of any null values, dataframe.isnull() function was used. The data were directly read from the CSV files as training and testing data. After reading data from CSV using pandas.read_csv function. The data was stored as feather files for easier reading access for future use. The use of Pandas read_csv and concat to merge individual dataframes caused major bottlenecks by filling up ram. Feather is a language-agnostic data frame storage for python and R.[1]
Training and validation data were split in a ratio of 0.8:0.2 using sklearn's train_test_split function.

```
def process_data(data):
    import warnings
    with warnings.catch_warnings():
        warnings.simplefilter(action='ignore',
        ↪  category=FutureWarning)

        d1=data.iloc[:,0:1080]
        # print("D1",d1)
        d2=data.iloc[:,1080:1092]
        # print("\n D2",d2)
        d3=data.iloc[:,1092:1094]
        print("d1 before processing",d1.shape)
        print("d2 before processing",d2.shape)
        d1=d1.groupby((np.arange(len(d1.columns)) // 120),
        ↪  axis=1).mean()

        d2.insert(loc=4,column='Final_goal_q', value=
        ↪  (d2['Final_goal_qr'] + d2['Final_goal_qk']))
        d2.insert(loc=9,column='Local_goal_q', value=
        ↪  (d2['Local_goal_qr'] + d2['Local_goal_qk']))
```

```
        d2.insert(loc=12,column='Robot_pos_q', value=
        ↪  (d2['Robot_pos_qr'] + d2['Robot_pos_qk']))
        d2=d2.drop(['Final_goal_qr','Final_goal_qk'],
        ↪  axis=1,inplace=False)
        d2=d2.drop(['Local_goal_qr','Local_goal_qk'],
        ↪  axis=1,inplace=False)
        d2=d2.drop(['Robot_pos_qr','Robot_pos_qk'],
        ↪  axis=1,inplace=False)
        print("d1 after processing",d1.shape)
        print("d2 after processing",d2.shape)
        heading_1=list(d1)
        heading_2=list(d2)
        heading_1=heading_1+heading_2
        # print(d1)
        # print(d2)
        d1=pd.concat([d1,d2],axis=1,ignore_index=True)
        # print(d1.shape)
        return d1,d3,heading_1
```

A view of 30 degrees was considered as a single feature, which brought down the total columns of the laser range data from 1080 to 9 columns, thus, giving 9 features. Additionally, the quaternion ($q_r$andq_k) for the current robot position, local and final goal was summed up as a single variable q. This has resulted in a total of 18 columns as the input data features.

Chapter 3

MODEL SELECTION AND VALIDATION

After the completion of data processing, the data is then trained using the training data set and then the values are predicted using the validation dataset. Finally, the model with the lowest measure of error is chosen with confidence to be the better model and is used for the rest of the project.

Here, E_val is taken as the error measure for choosing a model over the other proposed models.E_val acts as a proxy for E_out. The model with the lower E_Val is chosen to continue further tuning of hyperparameters and regularization. Using the learning_curves function from the sk_learn package, there is no necessity of needing to pass a validation dataset to it as it automatically does it.

3.1 Linear Regression

```
 # Importing and fitting data for linear regression
lin_reg=LinearRegression()
lin_reg.fit(x_n_train,y_n_train)
#predicting for val dataset and calculating RMS error
pred=lin_reg.predict(x_n_val)
error=mean_squared_error(y_n_val,pred[:,:],multioutput=
↪  'raw_values')
print(error)
#plotting learning Curve
from sklearn.model_selection import learning_curve
import numpy as np
import matplotlib.pyplot as plt
training_sizes, training_scores,validation_scores =
↪  learning_curve(
    estimator = LinearRegression(),
    X = x_training_data,
    y = y_training_data,
    train_sizes = np.linspace(5, len(x_training_data) * 0.8,
    ↪  dtype = int) ,
    cv = 5,
```

```
    scoring = 'neg_mean_squared_error'
)
#plotting training and validation curves
line1 = plt.plot(
    training_sizes, -validation_scores.mean(axis = 1),'b')

line2 = plt.plot(
    training_sizes, -training_scores.mean(axis = 1),'r')
```

Using linear regression, we can see that we get an of error about [0.11376724 0.05841072] overall for the predicted values of y of the validation dataset.

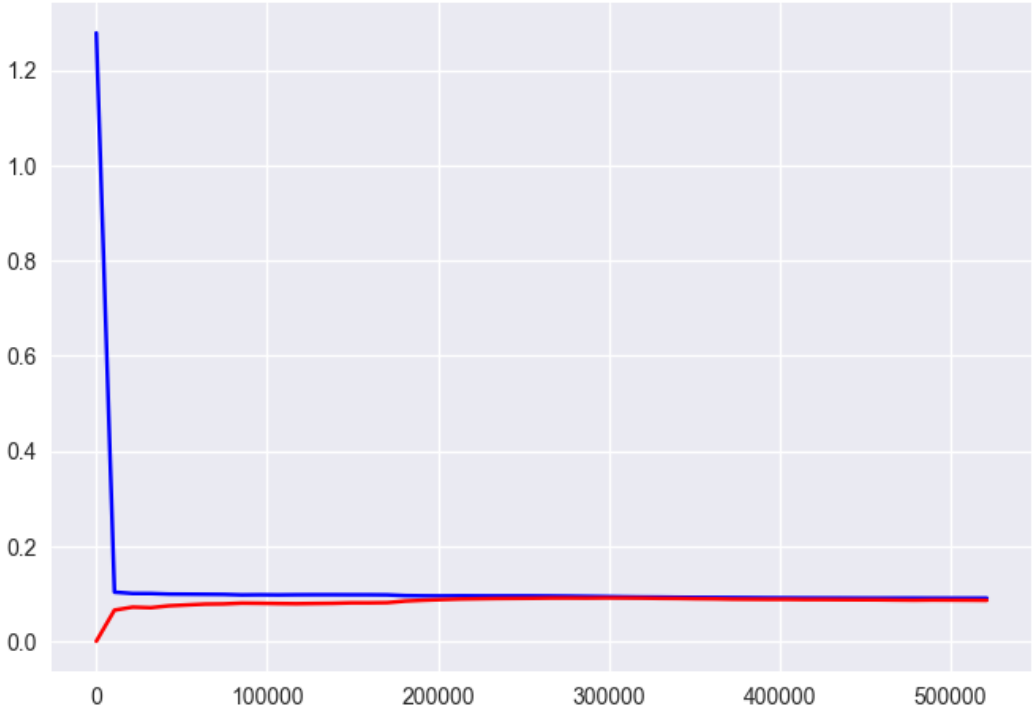

Figure 3.1: Plot of Training and Validation Curves, Error vs Amount of Data

From Fig 3.1, We can see that the curves meet, indicating that adding more points to the dataset will not improve the error in any way. The red line is the training set while the blue line is the validation set. Linear regression is not the best algorithm as the error is not zero, but it is about on average 0.1 throughout the plot. The validation set had an error that fell from 1.2 to about 0.1 after about 10000 data points.

3.2 Decision Tree Regression

```
#Import decision tree and fitting training dataset
from sklearn.tree import DecisionTreeRegressor
model = DecisionTreeRegressor()
model.fit(x_n_train,y_n_train)
pred=model.predict(x_n_val)
error=mean_squared_error(y_n_val,pred[:,:],multioutput=
↪  'raw_values')
print(error)
#plotting learning curve
from sklearn.model_selection import learning_curve
import numpy as np
import matplotlib.pyplot as plt
training_sizes, training_scores,validation_scores =
↪  learning_curve(
    estimator = DecisionTreeRegressor(max_depth=10),
    X = x_training_data,
    y = y_training_data,
    train_sizes = np.linspace(5, len(x_training_data) * 0.8,
    ↪  dtype = int),
    cv = 5,
    scoring = 'neg_mean_squared_error'

)
# plotting training and validation curves
line1 = plt.plot(
    training_sizes, -validation_scores.mean(axis = 1), 'b')

line2 = plt.plot(
    training_sizes, -training_scores.mean(axis = 1), 'r')
    #decision tree algorithm with max_depth=10
from sklearn.tree import DecisionTreeRegressor
model = DecisionTreeRegressor(max_depth=10)
model.fit(x_n_train,y_n_train)
pred=model.predict(x_n_val)
error=mean_squared_error(y_n_val,pred[:,:],multioutput=
↪  'raw_values')
```

```
print(error)
```

For no max_depth defined, the model defaults to expanding all nodes till the leaves get pure. Here we run the model for max_depth = 10 and 100 once again. When max_depth is not defined, we get an E_val of [0.00421088 0.00879876], which is very low. The data being passed has not been standardized.
As seen in fig 3.2, the left side plot is the training curve, while the right side plot is the validation curve. But on analyzing the graphs, we can see that the model overfits.

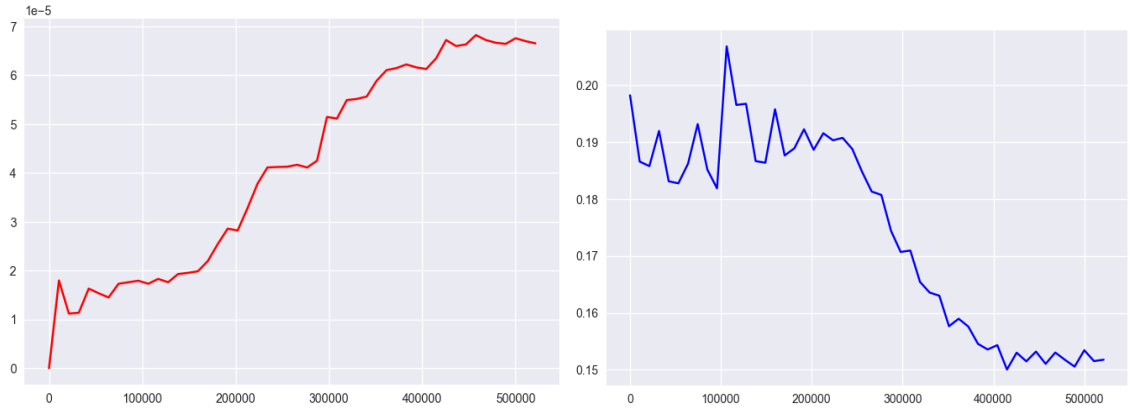

Figure 3.2: Plots of the Model

When the max_depth is set to about 10, we get an E_val of [0.05269766 0.05242499].From Fig 3.3, we can see that training and validation curves are much better and that they can merge if more data points can be given.

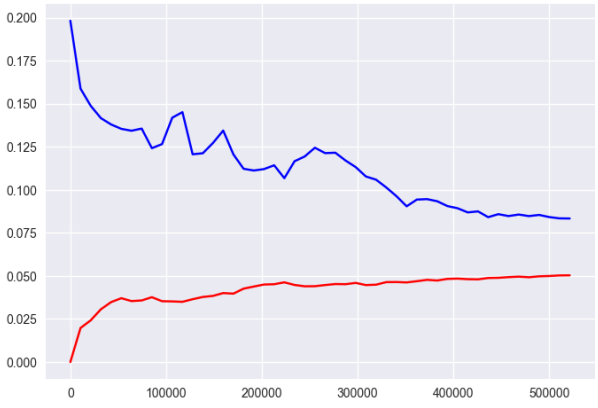

Figure 3.3: Plot of Training and Validation Curves for depth about 10

When the max_depth is set to about 100, we get an E_val of [0.00415528 0.00891209].

From Fig 3.4, we can see that training and validation curves overfit by a greater margin compared to the previous plot.

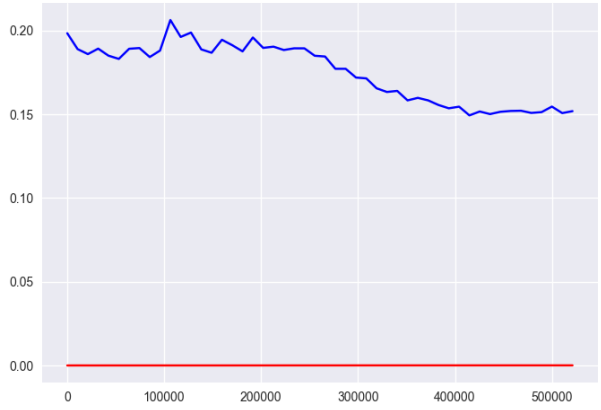

Figure 3.4: Plot of Training and Validation Curves for depth about 100

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
standardized_data = scaler.fit_transform(x_training_data)
```

When we scale the data using StandardScaler from sklearn, we run it for max_depth=10 and we can see that the results we get seem error values of [0.05628819 0.05337336]. The graph has not changed much but there has been a slight increase in error compared to running the same model on non-standardized data.

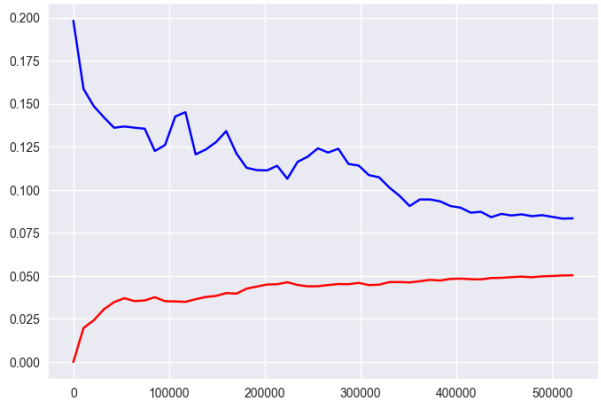

Figure 3.5: Plot for depth=10 and standardized data

After plotting different learning curves, the max_depth is taken as 10 for baseline tuning the parameters.

3.3 Neural Networks

```
#setting up the neural network from keras
import tensorflow
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense

#setting up layers
neural = Sequential([
    Dense(18, activation='tanh'),
    Dense(18, activation='tanh'),
    Dense(2, activation='sigmoid')
])
#compiling the neural network
neural.compile(optimizer='sgd',
               loss='mean_squared_error',
              metrics=['mean_squared_error'])


hist = neural.fit(x_n_train, y_n_train, batch_size=1000,
↪  epochs=50,validation_data=(x_n_val, y_n_val))

#predicting values
pred1=neural.predict(x_n_val)

#finding error
error_1=mean_squared_error(y_n_val.iloc[:,0:1],pred1[:,0],
↪  multioutput='raw_values')
print('Eval of v',error_1)
error_2=mean_squared_error(y_n_val.iloc[:,1:2],pred1[:,1],
↪  multioutput='raw_values')
print('Eval of w',error_2)

#learning curve
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.ylabel('Loss')
plt.xlabel('Epoch')
```

```
35 plt.legend(['Train', 'Val'], loc='upper right')
36
37 plt.show()
38
```

The type of neural network used here is a Keras Sequential and uses the Dense library to build upon the layers of the model. There are 3 layers,2 Dense layers, and an Output layer with 18,18 and 2 hidden layers for each layer of the network. The hidden layer values were determined after a few rounds of hit and trial. The activation functions used are tanh and sigmoid. The optimizer used here is Stochastic Gradient Descent and loss function as mean squared error.[2][3] The network is compiled after various required parameters have been defined for the system. The model has been trained for 50 epochs.

We get E_val as [0.07628705 0.06021202] which is a bit higher than Decision Tree Regression but lower than linear Regression.

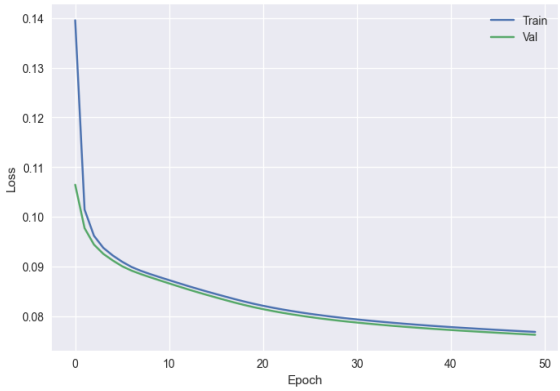

Figure 3.6: Plot of Epochs vs Error

In fig 3.6, it is seen that the model has a pretty good fit, but getting a good fit takes time as many parameters and layers have to be tuned.

3.4 The Chosen Model

Decision Tree Regression was chosen as the final model of choice as the E_val value was the lowest when max_depths was regulated. This model also allows for easier tuning of hyperparameters compared to Neural Networks and is simpler to regulate the model to reduce overfitting. Alternatively, neural networks can be used with more tuning as they also offer good performance.

Chapter 4

REGULARIZATION AND HYPERPARAMETER TUNING

4.1 Hyperparameters

For the decision tree regressor, the following are the hyperparameters[4]:

- Max_Depth: This determines the max depth of the tree. If this is high, it will most certainly cause overfitting.
- min_samples_split: The minimum number of samples an internal node must have before it can be split into more nodes.
- min_samples_leaf: This governs the external node must have before it can give output
- max_feature: This determines the max features the model should consider from the given input model. By default, it considers all features.

For each hyperparameter, validation curves [5] have been plotted using the yellow-brick package [6]. This package allows for efficiently controlling a single parameter and seeing how the model reacts.

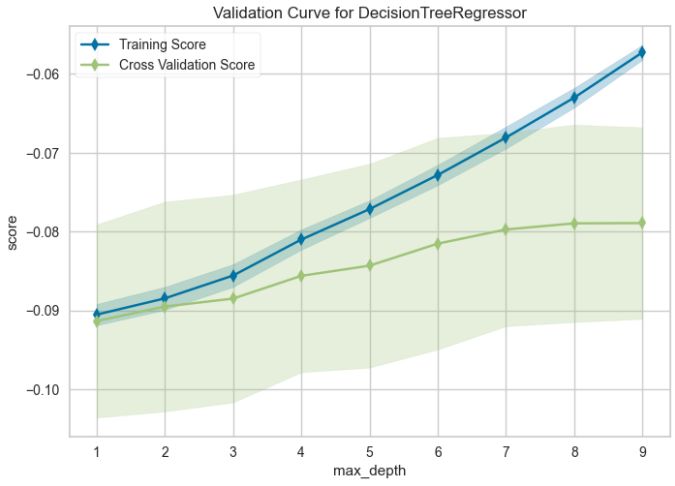

Figure 4.1: Max_Depth

When the Max_depth is varied from 1 to 10, we can see the least error at 1 and proportionally increases as the depth increases to 10. But the model under fits for

depth below 7 and overfits when depth is above 7, accordingly affecting error values.

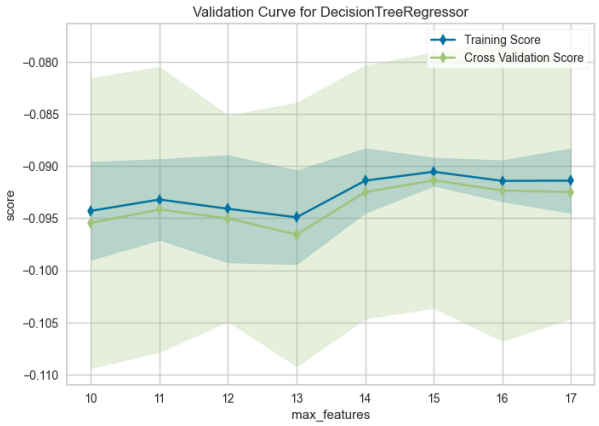

Figure 4.2: Max_Feature

Varying the number of max features does not affect the model much as the gap between the training and validation curves was constant.

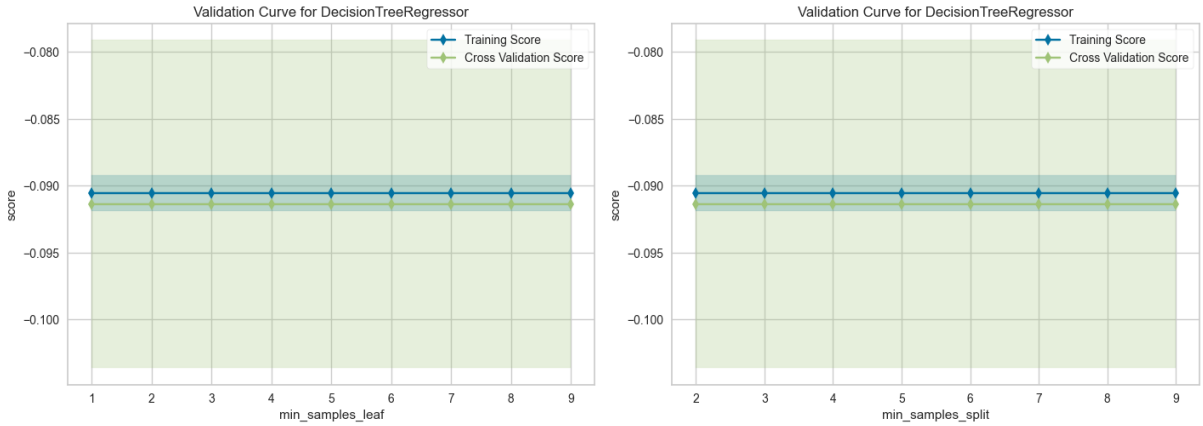

Figure 4.3: min sample leaf and min sample split

Varying the parameters of the min sample leaf and the min sample split does not affect the model much as the lines are pretty much constant and in a straight line.

4.2 Regularization

Regularization for decision trees is taken in the general sense as any method to prevent the over-fitting of training data. Regularization is done by tuning and controlling the hyperparameters. For this project, it has been decided to regularize the max depth of the decision tree regressor as that offers noticeable results when

tuning hyperparameters for this data set.

As we can see from the validation graphs of hyperparameter tuning, change in max_features and max_depth change, but in max_features, there is no change in the gap between the two curves. So the max_depths is considered as the parameter to be regularized. Overfitting in decision tree regressor is controlled by the depth of the tree, as shown in the previous section, when the depth was 10, the fit was better than the depth at 100. From fig 4.1, we can see that the fit we get is better when the depth is at 7. Hence, the learning curve for depth = 7 has been calculated with an error(E_val) value of [0.09062949 0.06737162].

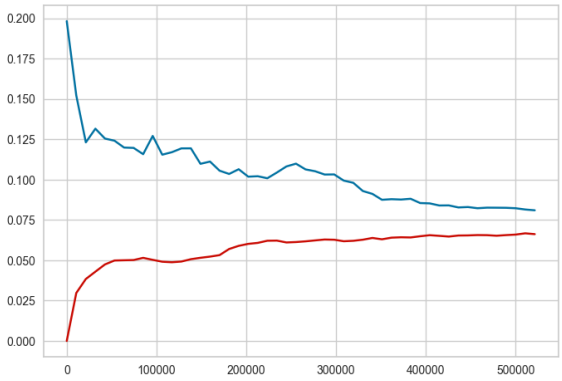

Figure 4.4: Learning Curve for depth=7

From Fig 4.4, we can see that the training and validation curve almost merges, more data points would allow them to converge.

Chapter 5

CONCLUSION

The best model for this dataset has been chosen as Decision Tree Regression with a max_depth parameter of 7. The next closest alternative would be choosing a neural network with 3 layers with the activation functions, tanh and sigmoid. The overall error (E_{out}&E_{Test}) of the model can be found by:

$$E_{out} = E_{Test} + \sqrt{\frac{8}{N} * \ln \frac{4*((2*N))^{d_{vc}})}{\delta}}$$

```
E_test for Y_test = [0.09062949 0.06737162]
E_out_v=  0.2774184977453761
E_out_w=  0.25416062729773226
```

Figure 5.1: E_out and E_test

For future work, the models could be run on the full dataset and have their performance differ from the ones presented. Hence, a suitable machine learning model for the given problem statement has been delivered.

REFERENCES

[1] Apache Software Foundation. *Feather file format*. https://arrow.apache.org/docs/python/feather.htmlr. [Online]. 2022.

[2] Machine Learning Mastery. *Neural Networks for deep learning*. https://machinelearningmastery.com/tensorflow-tutorial-deep-learning-with-tf-keras/. [Online]. 2021.

[3] Tensorflow. *Keras Tutorials*. https://www.tensorflow.org/tutorials/keras/regression. [Online]. 2021.

[4] ken-hoffman. *Hyperparameters of Decision Tree Explained*. https://ken-hoffman.medium.com/decision-tree-hyperparameters-explained-49158ee1268e. [Online]. 2020.

[5] Rukshan Pramoditha. *Valdiation Curve Explained*. https://towardsdatascience.com/validation-curve-explained-plot-the-influence-of-a-single-hyperparameter-1ac4864deaf8. [Online]. 2021.

[6] scikit-yb developers. *Validation Curve*. https://www.scikit-yb.org/en/latest/api/model_selection/validation_curve.html. [Online]. 2022.