Pipeline Used

Data Cleaning

Data cleaning is the process of preparing data for analysis by removing or modifying data that is incorrect, incomplete, irrelevant, duplicated, or improperly formatted. It helps in maintaining quality and making for more accurate analytics, which increases effective, intelligent decision-making. In my project, I have used 2 data cleaning functions. One that removes the duplicate rows from the pandas dataframe and second which removes the rows that have any element as NaN (not a number).

Feature Selection

I have selected features which I think are making an impact on the rotational and translational velocity of the bot. The features selected are shown below.

1. Distance between current position and local goal
2. Distance between current position and final goal
3. Distance between local goal and final goal
4. Angle of line-of-sight (LOS) joining current position and final goal with the x axis
5. Angle of line-of-sight (LOS) joining current position and local goal with the x axis
6. Distance in front of the bot (540th value in the laser range)

More features can also be used to have a better prediction but what I noticed from my results is that increasing features is not making much difference.

Various Models Considered

1. Linear regression - Ridge regularization
2. SVM
3. XGBoost

Data Splitting

Splitting the data means extracting various data from the input file. In our case, the file has laser range, final goal, local goal, the position of the bot and command actions. I extracted these columns from the csv file using array slicing after reading the file through pandas and converting it to a numpy array.

Regularization

Regularization refers to techniques that are used to calibrate machine learning models in order to minimize the adjusted loss function and prevent overfitting or underfitting. Using Regularization, we can fit our machine learning model appropriately on a given test set and hence reduce the errors in it. Regularization is very useful in the cases where the data is noisy. If there is noise in the training data, then the estimated weights won't generalize well to the future data. This is where regularization comes in and shrinks or regularizes these learned estimates towards zero. My usage of regularization is explained below along with the outputs for each model.

Hyperparameter Tuning

Hyperparameters are defined as the parameters that are explicitly defined by the user to control the learning process. Hyperparameters are used by the learning algorithm when it is learning but they are not part of the resulting model. At the end of the learning process, we have the trained model parameters which effectively is what we refer to as the model. The hyperparameters that were used during training are not part of this

model. We cannot for instance know what hyperparameter values were used to train a model from the model itself, we only know the model parameters that were learned.

Here are some common examples

- Train-test split ratio
- Learning rate in optimization algorithms (e.g. gradient descent)
- Choice of activation function in a neural network (nn) layer (e.g. Sigmoid, ReLU, Tanh)
- Number of iterations (epochs) in training a neural network

How I have tuned hyperparameters is explained below along with the outputs for each model.

Validation

In the training process, each dataset was split into training and validation dataset (80:20 split). Next, after training each model by tuning the hyperparameters, the model was tested on the validation dataset and R2 score was calculated. Next, the hyperparameter corresponding to the max R2 score was taken and the model was trained again using the training+validation dataset. This resulted in the final model that could be used to test any external dataset.

Training and Validation Curves

Outputs for each model are present in this document. All the curves in this document are for training and validation. No curves for testing have been plotted.

Evaluation Metrics

I have used the R2 score as my evaluation metric. It works by measuring the amount of variance in the predictions explained by the dataset. Simply put, it is the difference between the samples in the dataset and the predictions made by the model.

I thought of using the mean-squared-error and mean-absolute-error also but I felt they have an issue. If my data values are very small, then the error between any value and mean will also be very small. Squaring that error will make it smaller. So even if the R2 score was low, I was getting a small value of mean squared error. This is the reason why I did not include the MSE and MAE.

In the R2 score, we have the ratio of residual sum of squares (SSres) with the total sum of squares(SStot), so a small value of the sum of squares won't make any difference and it will be a good metric.

Conclusion - This was written after testing the models

The best model according to me is the XGBoost. The reason being that it gives a better performance as compared to the other 2. Also, it does not take a long time when the number of datapoints are high in the training.

All the models performed better when predicting the translational velocity (represented by cmd_vel_v in the document) as compared to the rotational velocity (represented by cmd_vel_w). The reason for this might be that the features selected by me do not contribute much to calculating the rotational component of the velocity.

NOTE: I have trained separate models for OpenBox and Corridor dataset. Since it was told in the class that testing and training should be performed on similar datasets. OpenBox and Corridor are different environments so training on one and testing on the other does not make any sense. Also, due to limited computational resources, I have used only a few files for training and testing.

Models and their Outputs

Linear Regression

I have used ridge regularization in my code. I am varying the regularization parameter (alpha or lambda) in a range from 1 to 10. For each value of lambda, I am calculating the R2 score of the actual output and predicted output. This was done for validation. Then I am taking the lambda corresponding to the highest R2 score and then using that lambda to train my model (by merging the train and validation set). After this my model is ready and can be tested on any external dataset.

Why regression?

- Regression analysis allows you to understand the strength of relationships between variables. Using statistical measurements like R-squared / adjusted R-squared, regression analysis can tell you how much of the total variability in the data is explained by your model.
- Regression analysis tells you what predictors in a model are statistically significant and which are not. In simpler terms, if you give a regression model 50 features, you can find out which features are good predictors for the target variable and which aren't.
- Regression analysis can give a confidence interval for each regression coefficient that it estimates. Not only can you estimate a single coefficient for each feature, but you can also get a range of coefficients with a level of confidence (eg. 99% confidence) that the coefficient is in.

Training files used for corridor:

- July22_1.csv
- July22_2.csv
- July22_4.csv

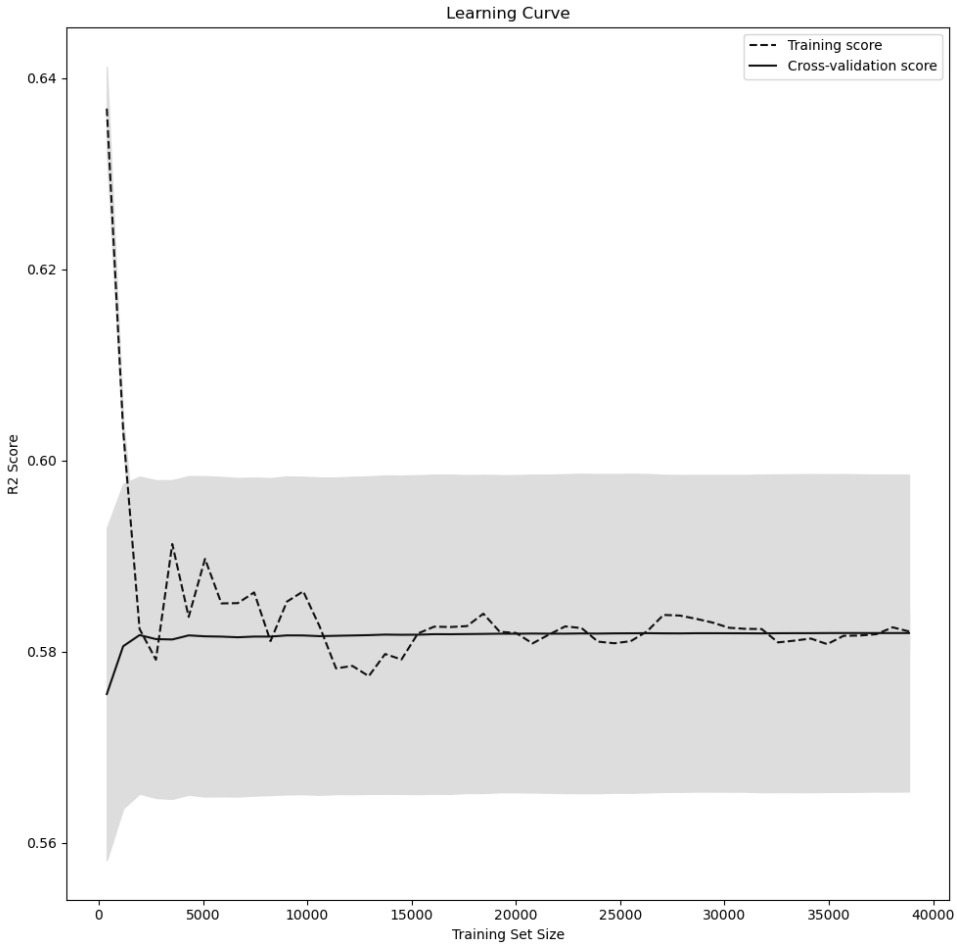
Test file used for corridor: July22_52.csv

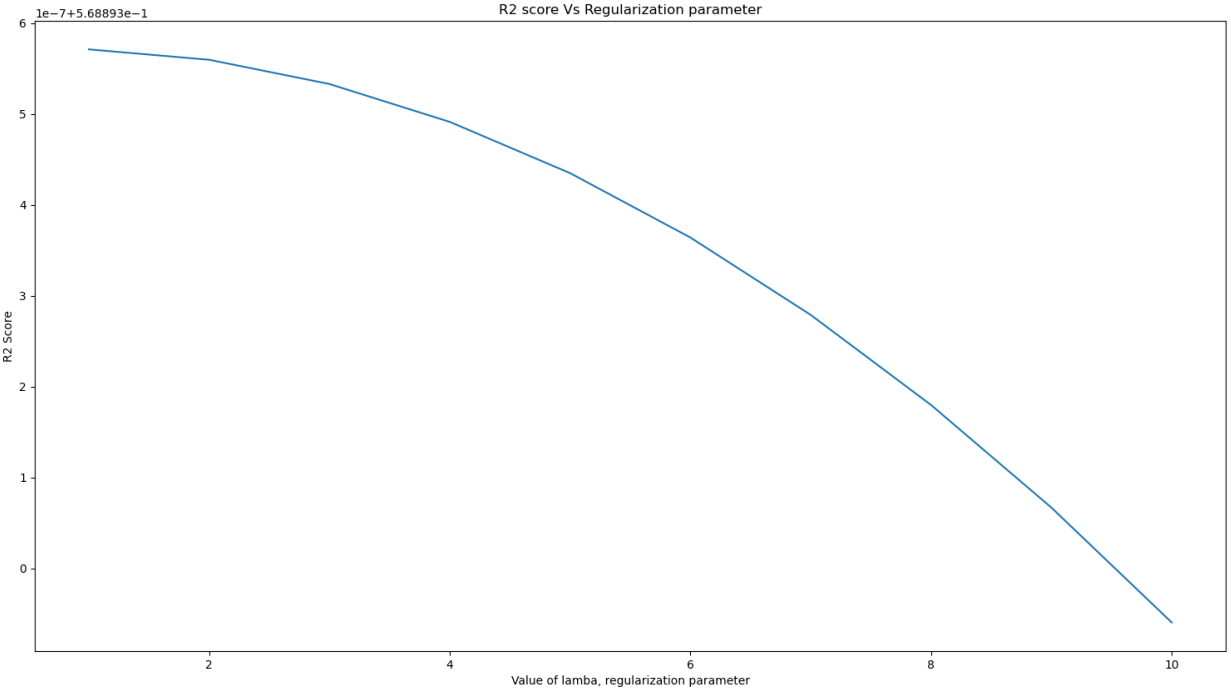
Training files used for open box:

- Aug14_Box_1.csv
- Aug14_Box_2.csv
- Aug14_Box_4.csv

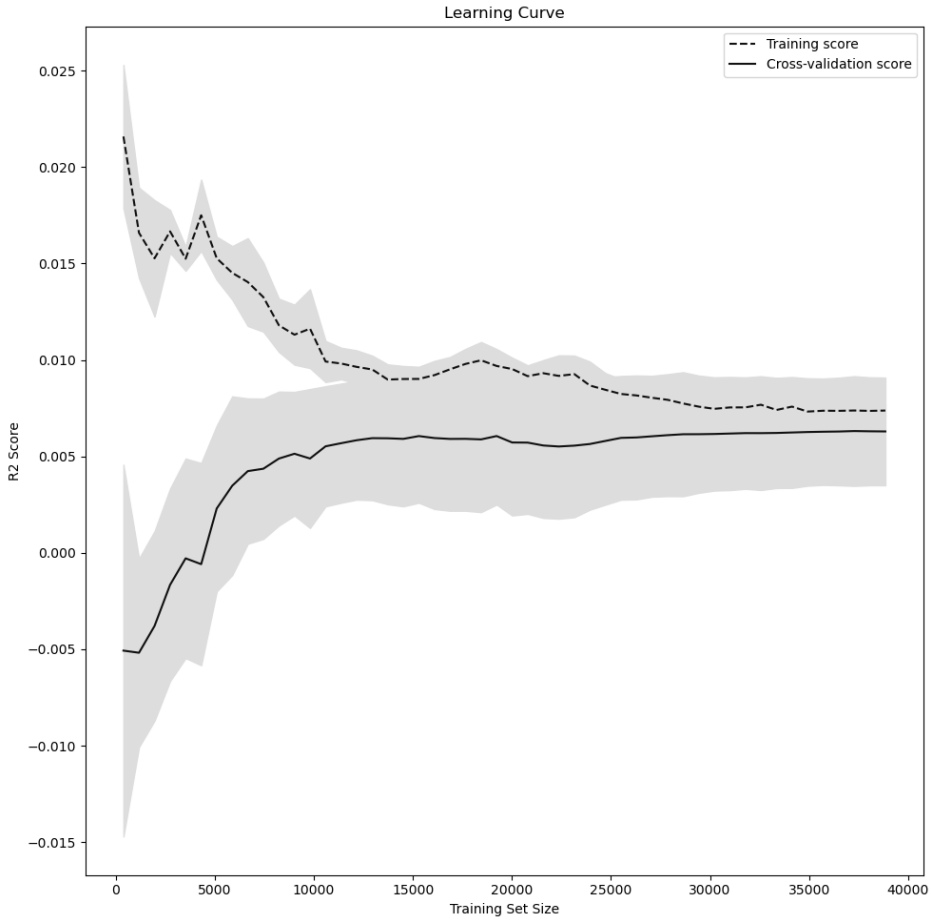
Test file used for open box: Aug14_Box_3.csv

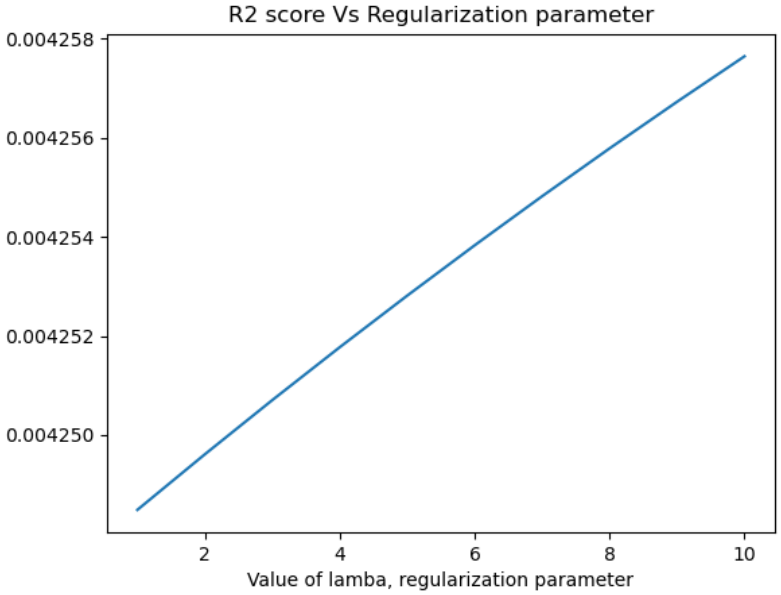
Data: OpenBox, lambda = 1 for the learning curve
For Cmd_vel_v
Lambda - 1 to 10
R2 scores for training list (varying lambda) - [0.5688935709183276, 0.5688935594675799, 0.5688935327793301, 0.568893491102288, 0.568893434712228, 0.5688933638326755, 0.5688932787144112, 0.5688931795958558, 0.5688930667111027, 0.568892940290004]
R2 score for the test data: 0.4704909958637611



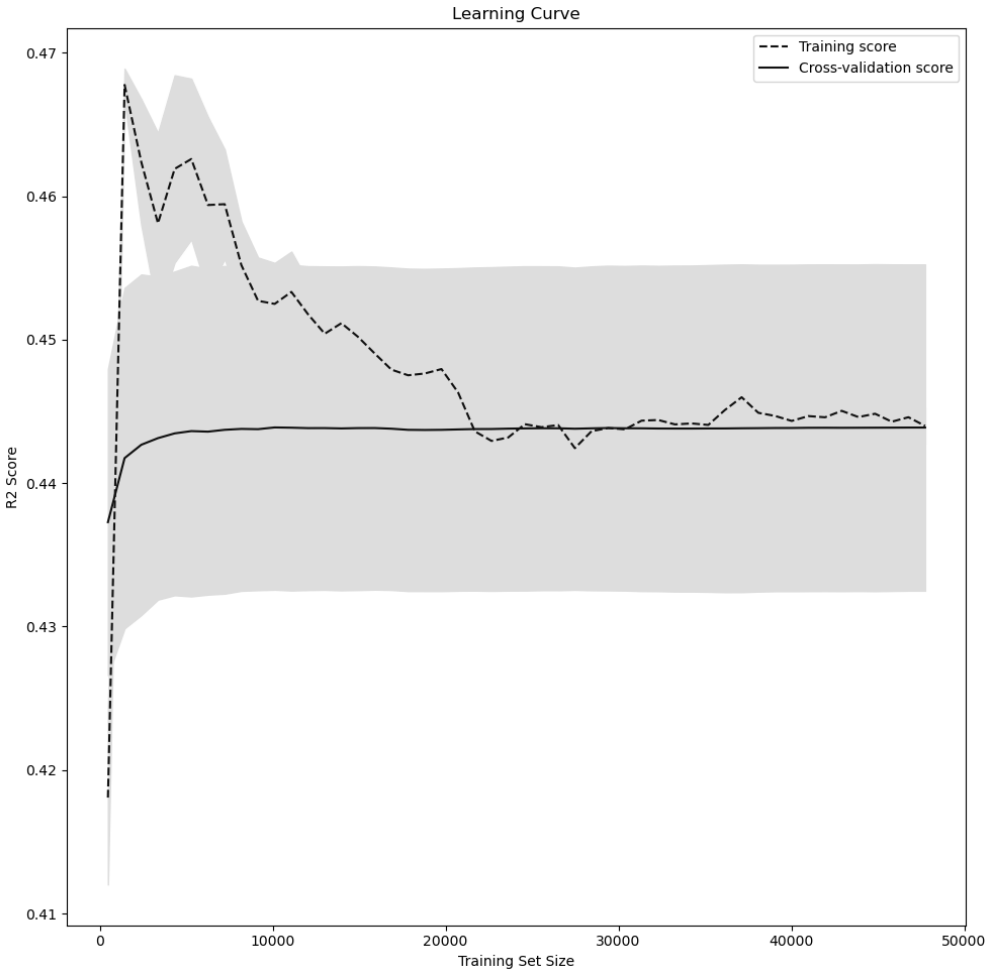


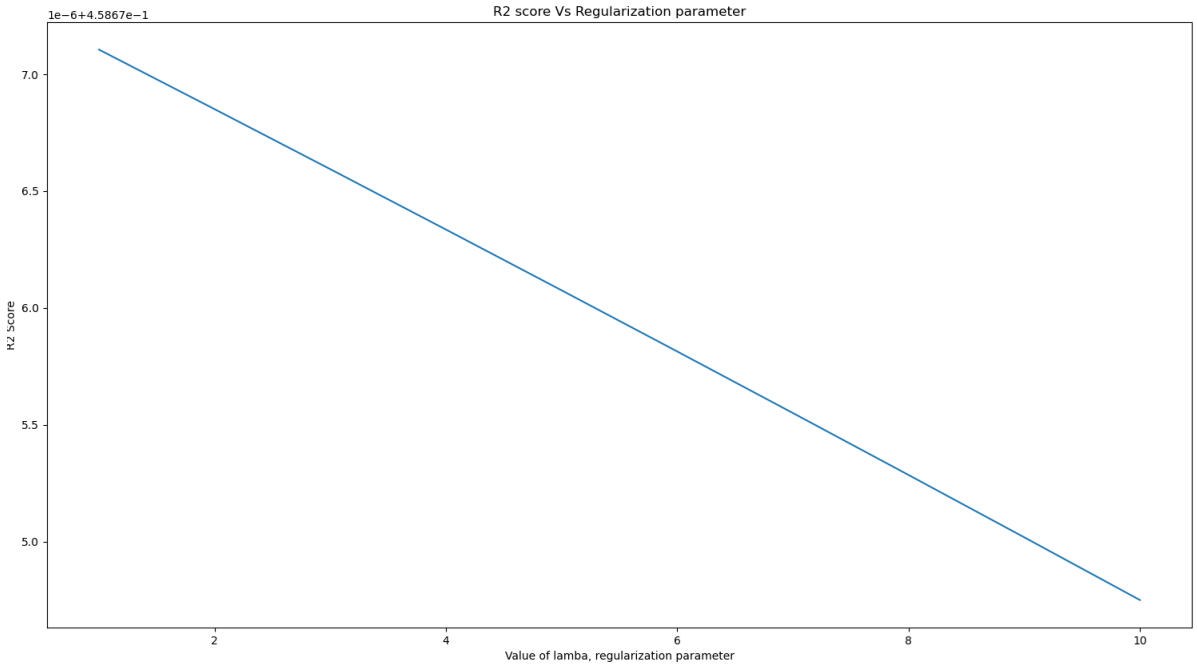
For Cmd_vel_w
Lambda - 1 to 10
R2 scores for training list (varying lambda) - [0.004248489674286482, 0.004249610608274557, 0.004250704532708704 5, 0.004251771902341384, 0.00425281316374726, 0.004253828755486966, 0.004254819108271524, 0.004255784645117111, 0.004256725781501935 5, 0.004257642925515892]
R2 score for the test data: -0.009242638445975837



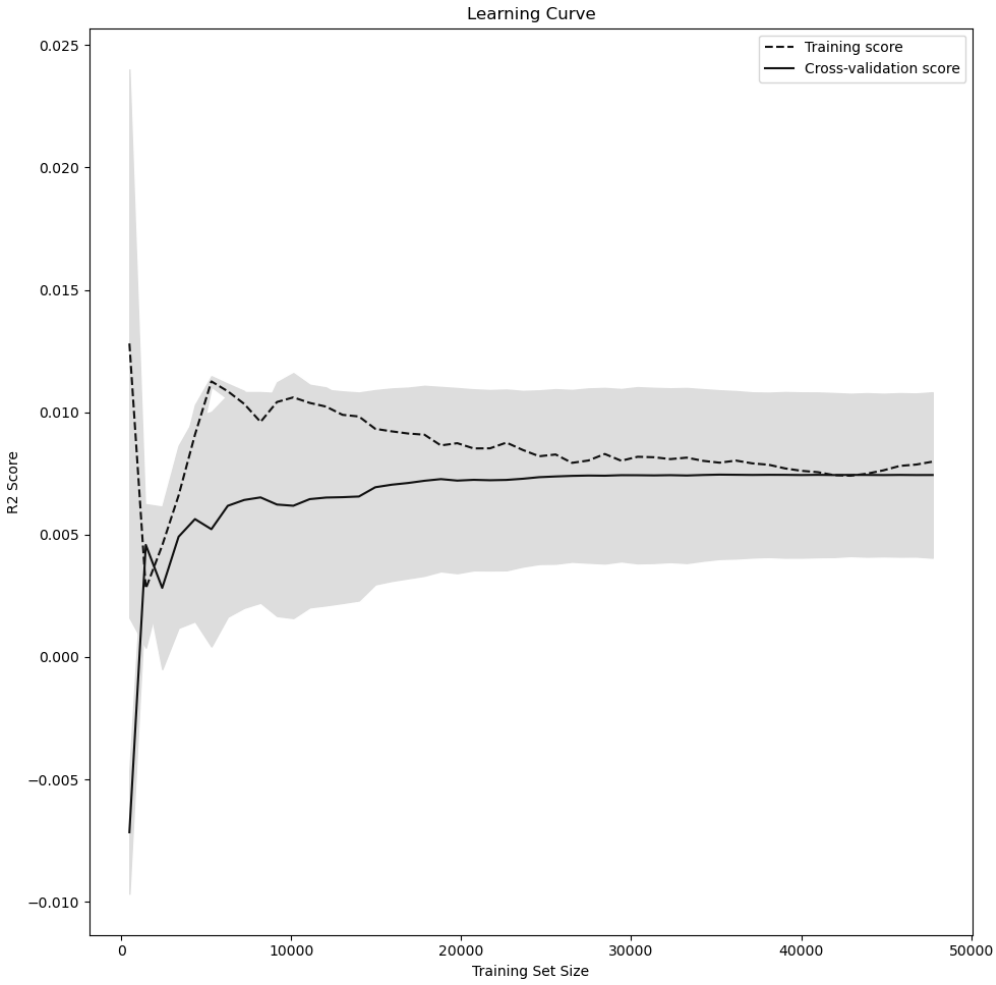


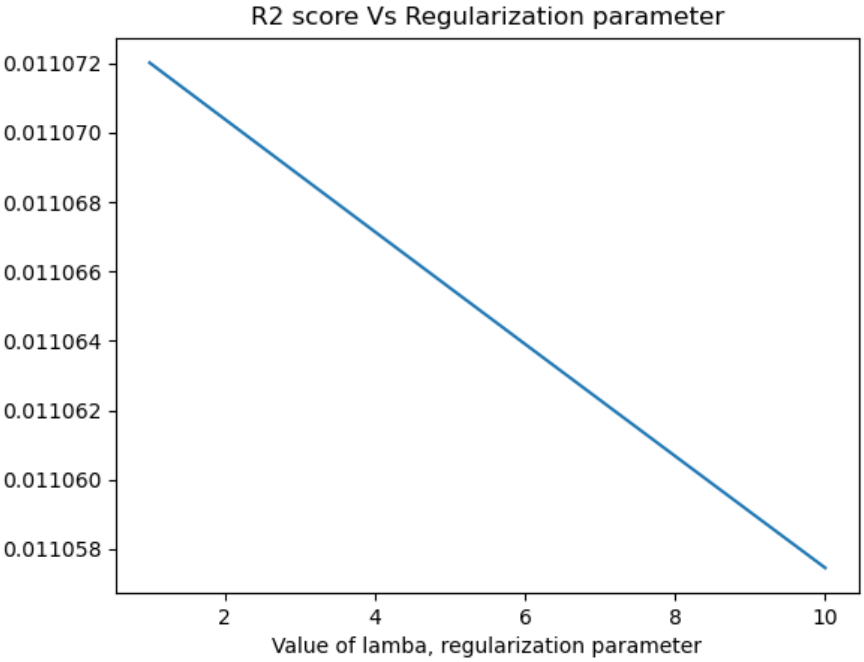
Data: Corridor, lambda = 1 for the learning curve
For Cmd_vel_v
Lambda - 1 to 10
R2 scores for training list (varying lambda) - [0.45867710555617847, 0.4586768505073332, 0.45867659376488257, 0.45867633533639474, 0.45867607522940435, 0.4586758134514132, 0.45867555000989046, 0.4586752849122724, 0.4586750181659627, 0.45867474977833333]
R2 score for the test data: 0.4924609831091059





For Cmd_vel_w
Lambda - 1 to 10
R2 scores for training list (varying lambda) - [0.011072011961764328, 0.011070387718418617, 0.011068765283261839, 0.011067144653927108, 0.011065525828045764, 0.011063908803248035, 0.011062293577163151, 0.011060680147418345, 0.011059068511640735, 0.011057458667456666]
R2 score for the test data: 0.0001576005514243306





SVM (Support Vector Machine)

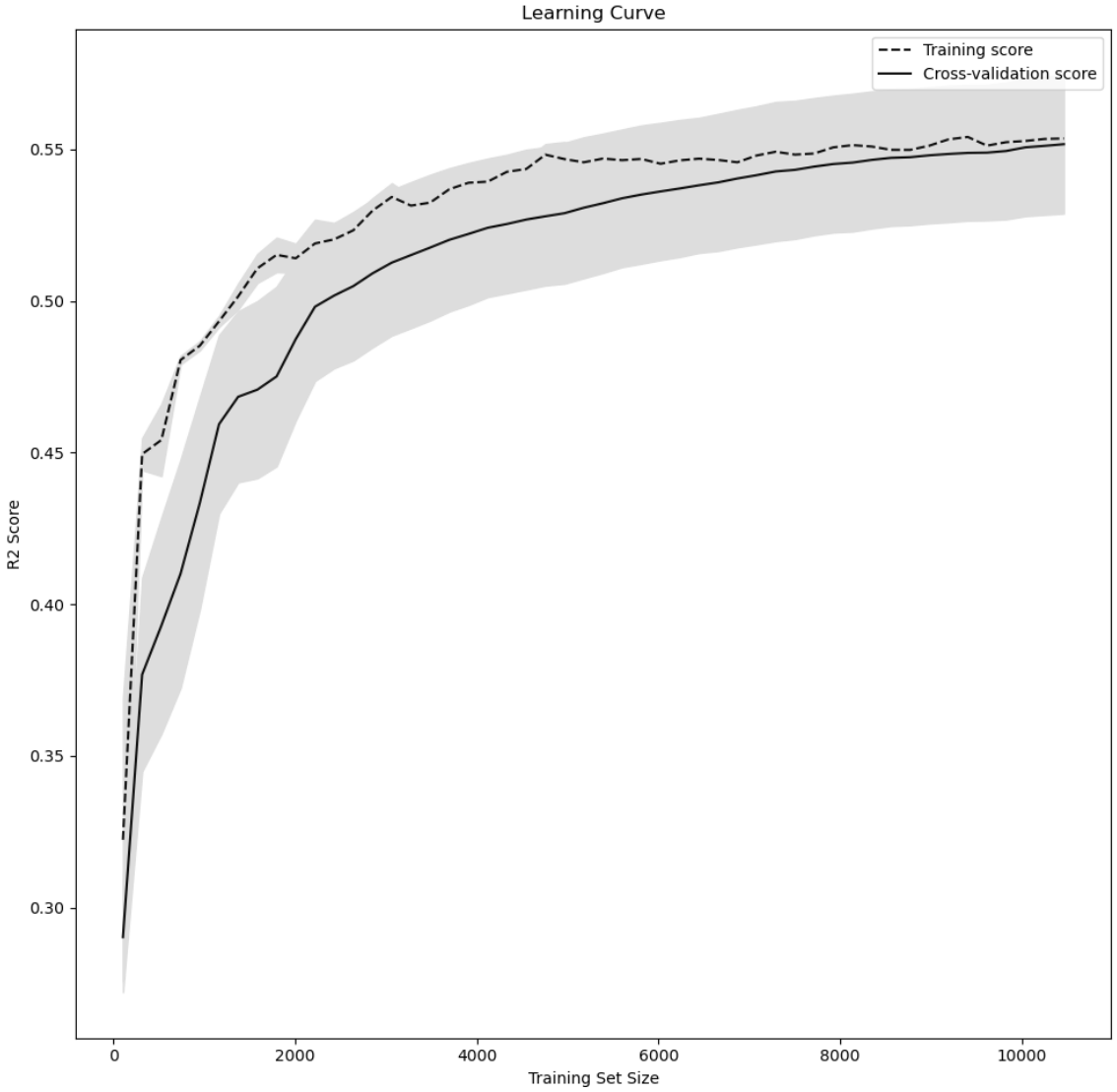
In SVM, I am modifying the regularization parameter lambda and then calculating the R2 score on the validation set for each lambda. Next, I am taking the lambda with maximum R2 score and using it to train the model again using the training+validation data set. I have also used 2 kernels in SVM, rbf and poly. I tried using sigmoid and linear also but they were taking too much time in the training. SVM can't be used on large datasets so I have trained the model using one input file (around 15000 points).

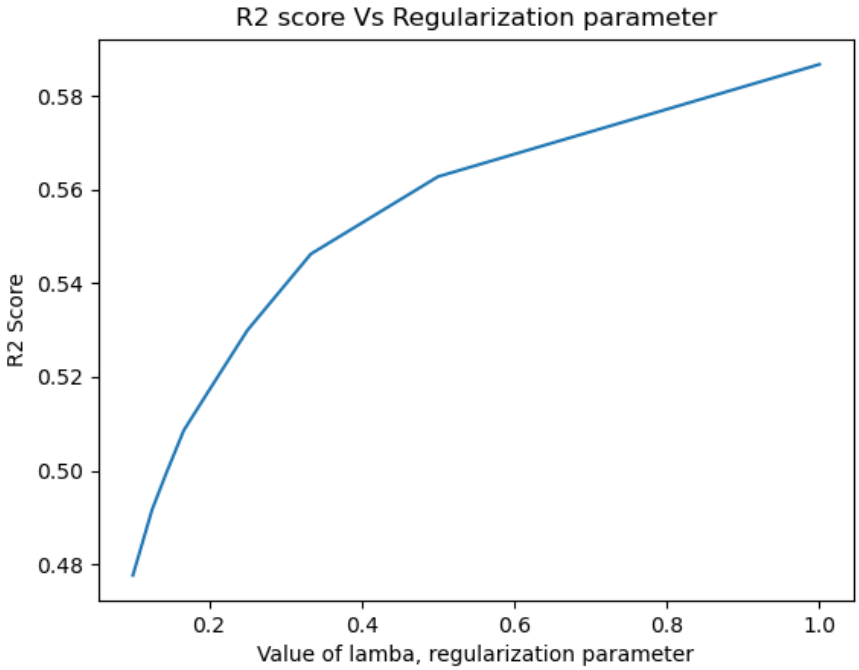
Why SVM?

- SVM works relatively well when there is a clear margin of separation between classes.
- SVM is more effective in high dimensional spaces.
- SVM is effective in cases where the number of dimensions is greater than the number of samples.
- SVM is relatively memory efficient

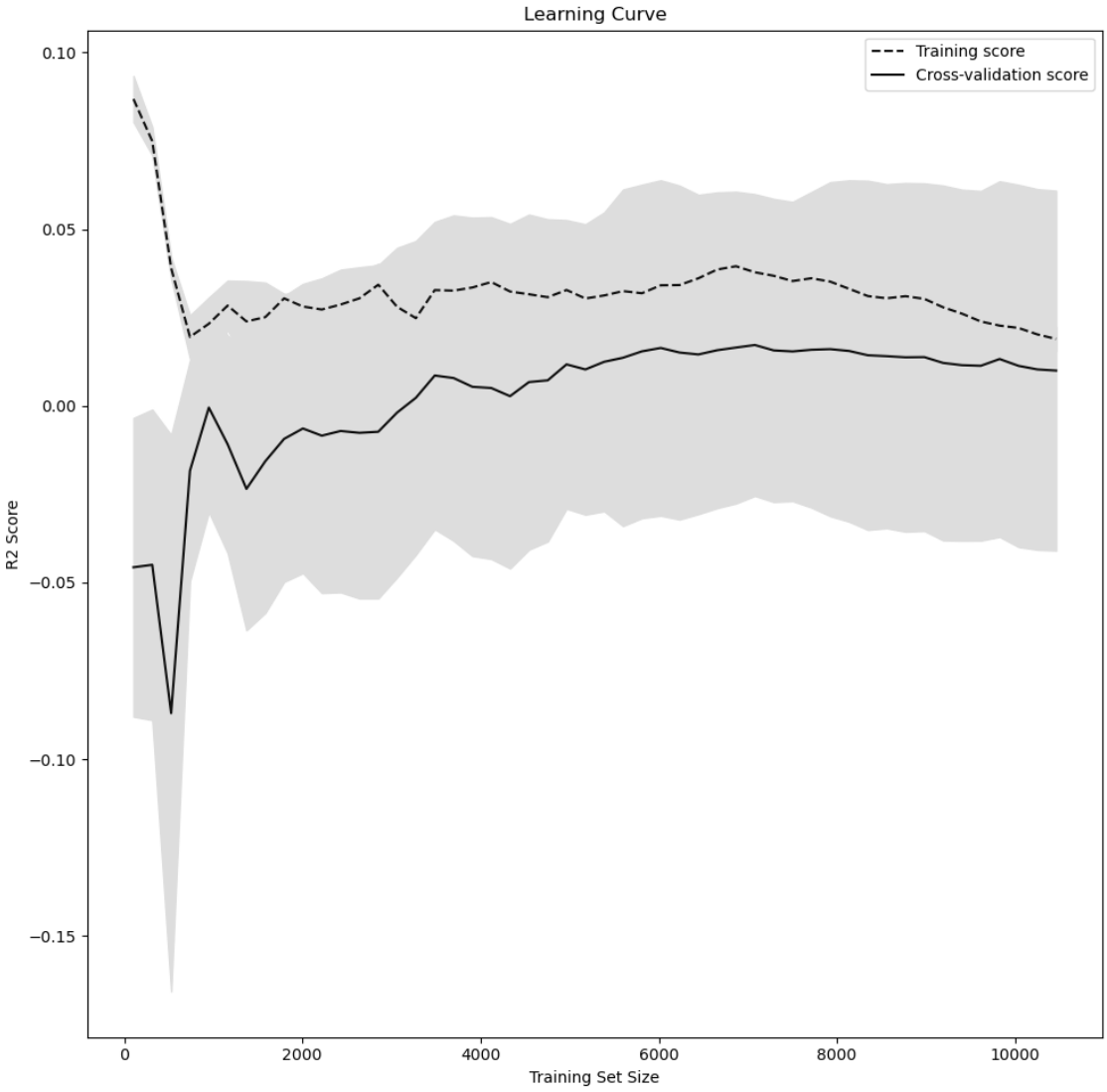
Data: OpenBox, lambda = 1 for the learning curve
Training file used: Aug14_Box_1.csv
Test file used: Aug14_Box_3.csv

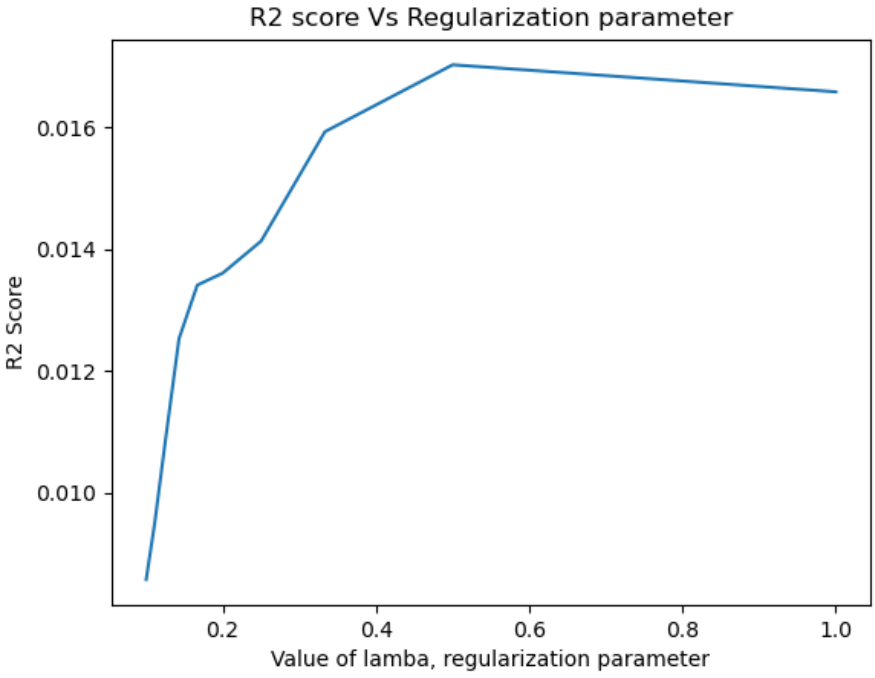
Kernel: Poly
For cmd_vel_v
R2 score: 0.4637854647432893



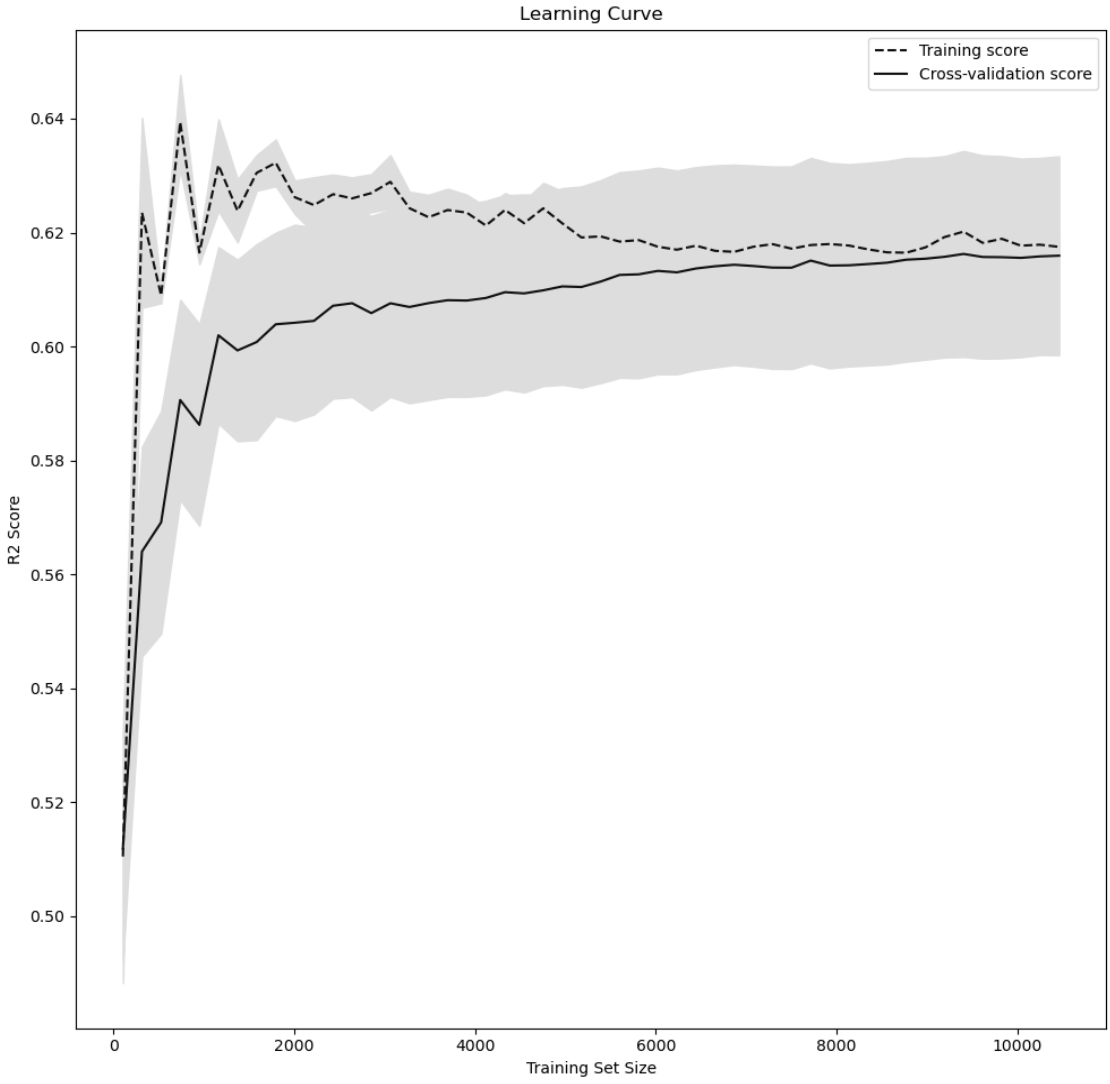


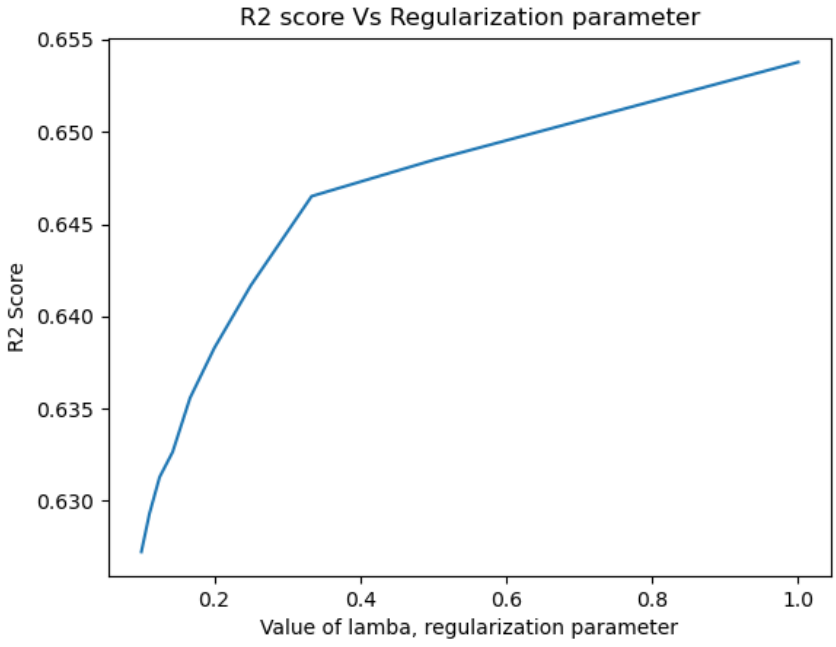
Kernel: Poly
For cmd_vel_w
R2 score: -0.27644861148703526



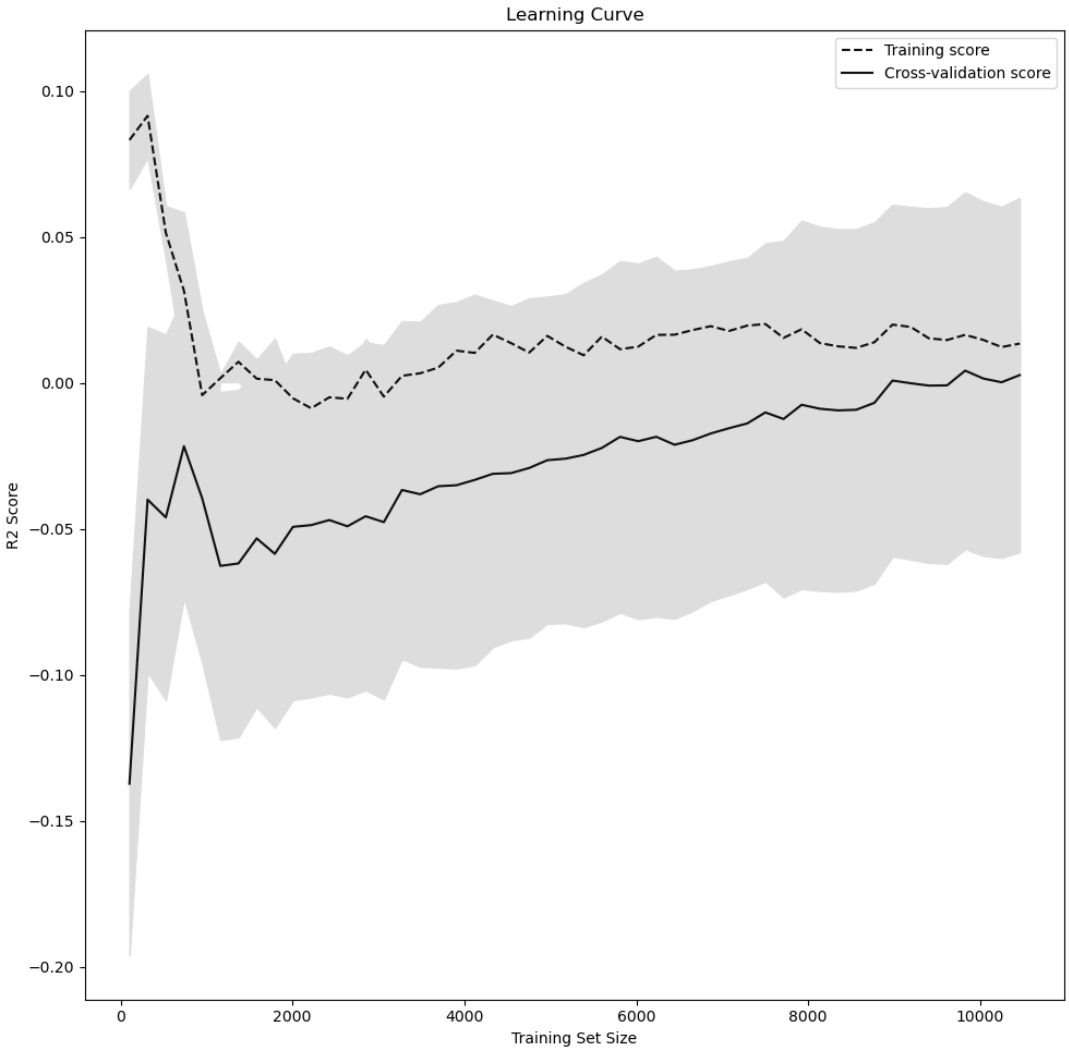


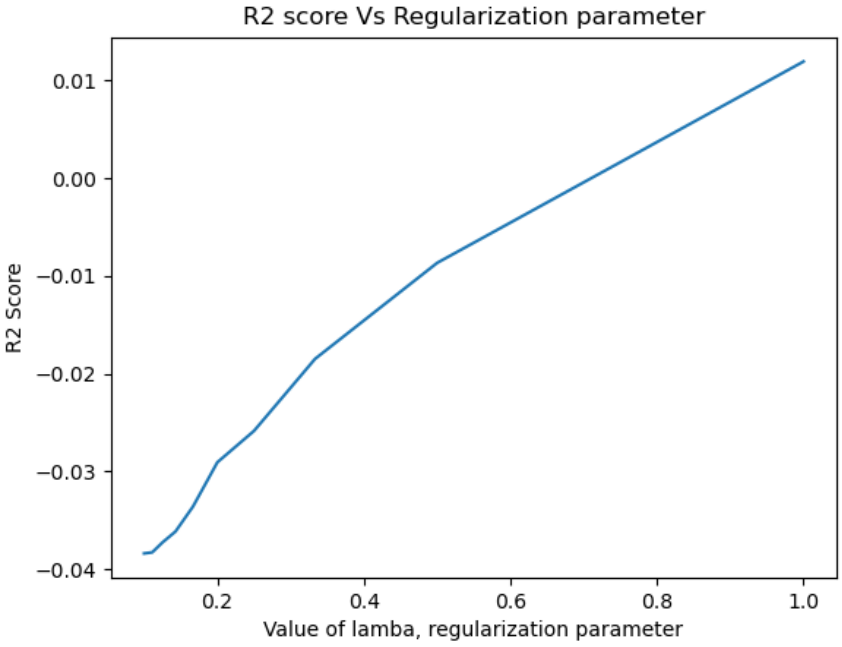
Kernel: rbf
For cmd_vel_v
R2 score for testing: 0.4637854647432893



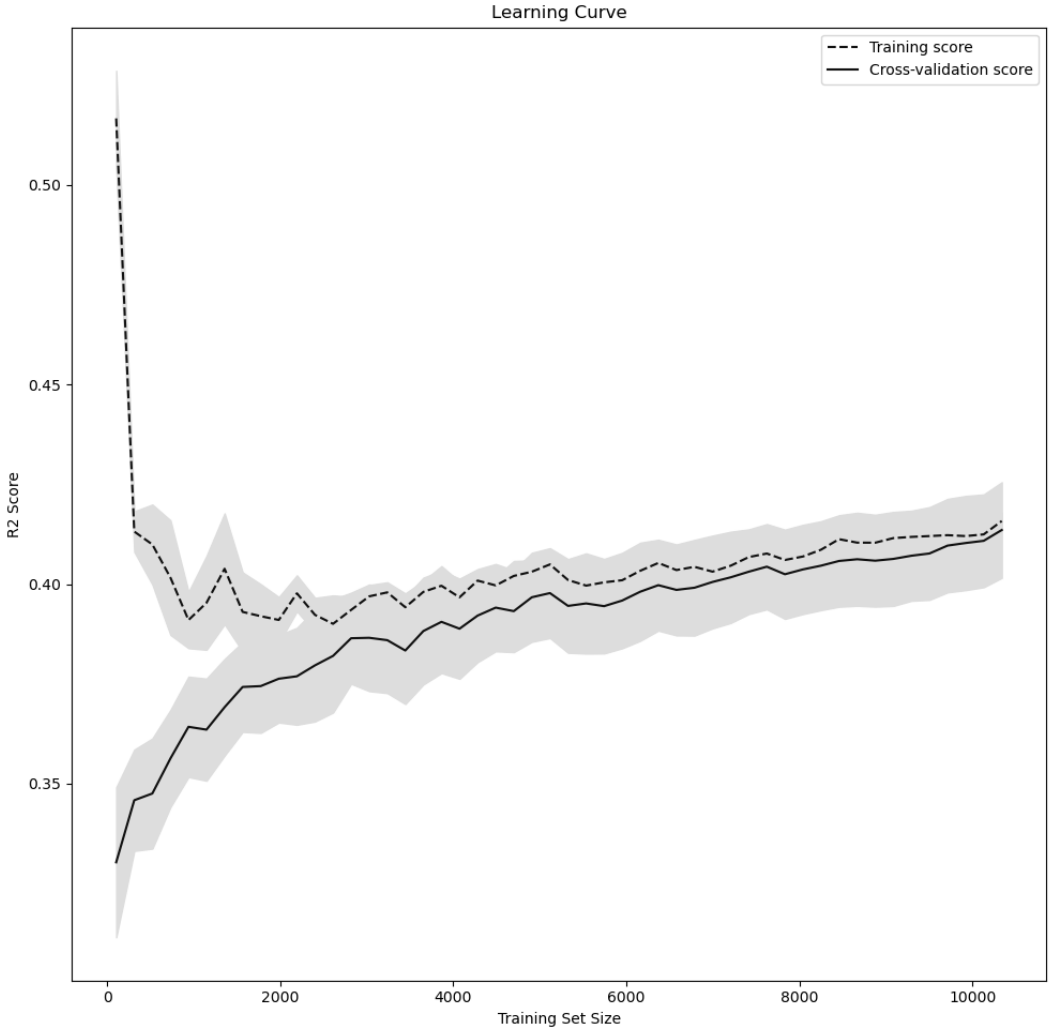


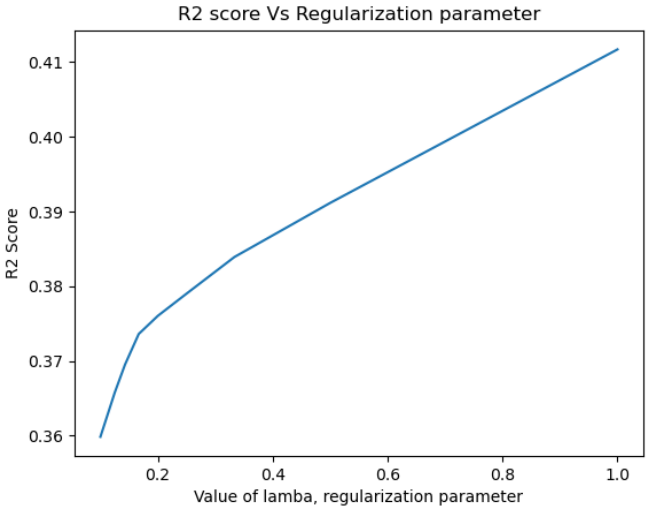
Kernel: rbf
For cmd_vel_w
R2 score of the test set: -0.2639969087804914



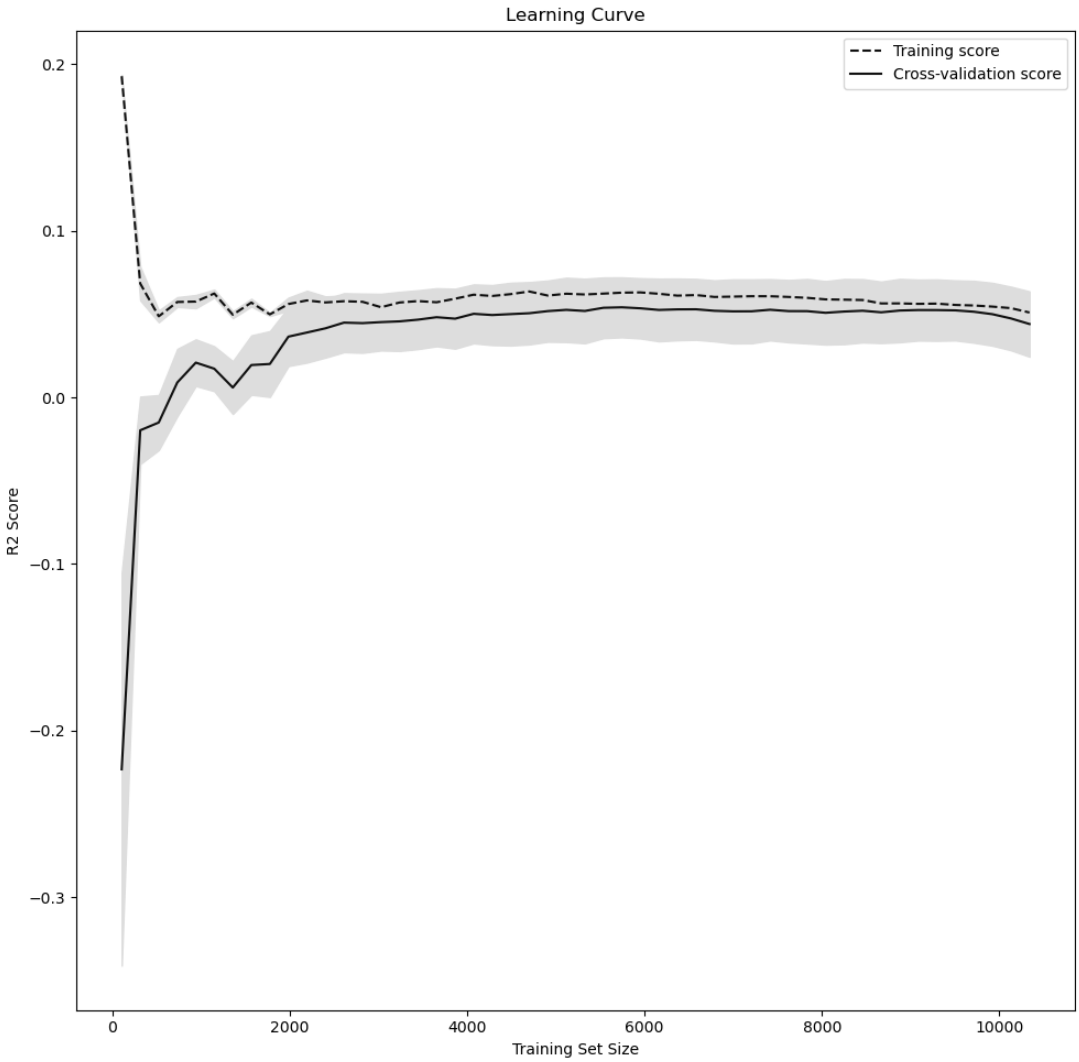


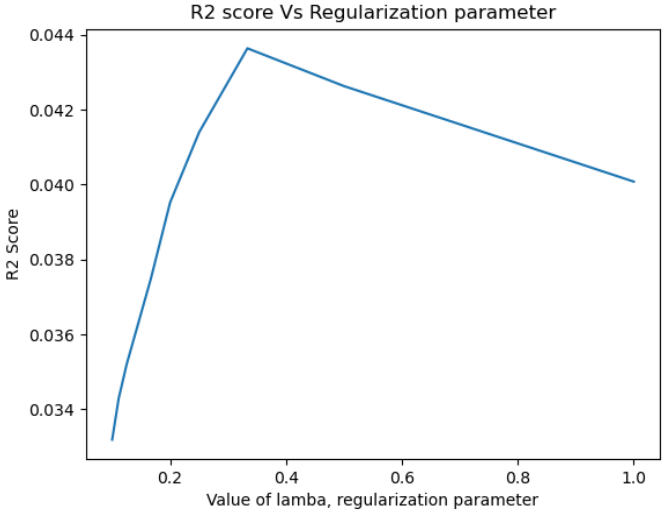
Data: Corridor, lambda = 1 for the learning curve
Training file used: July22_2.csv
Test file used: July22_52.csv
Kernel: rbf
For Cmd_vel_v
Lambda - 1 to 10
R2 scores for training list (varying lambda) - [0.4117055743613075, 0.39115520587244945, 0.38391346827870254, 0.3790098429460059, 0.37603332263554823, 0.37360083028142166, 0.36951158846568977, 0.3657693695500832, 0.3624677049673274, 0.35983527863345066]
R2 score for the test data: 0.4233913792092908





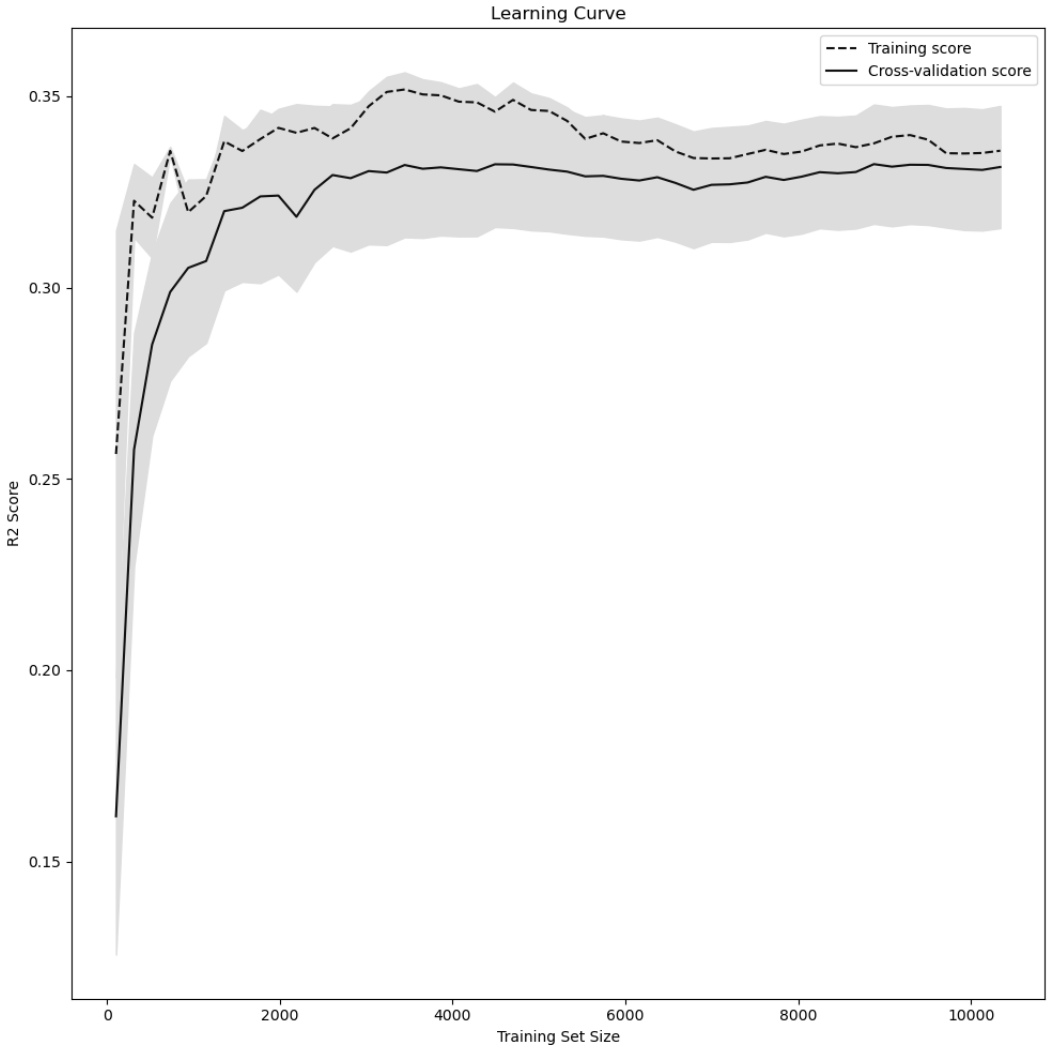
Kernel: rbf
For Cmd_vel_w
Lambda - 1 to 10
R2 scores for training list (varying lambda) - [0.04007873534230877, 0.042629321336758785, 0.0436378052145614, 0.04139581459183039, 0.03952340232496521, 0.03748186254526176, 0.03617861857895599, 0.035203038232863415, 0.03428270315059401, 0.03318999181889315]
R2 score for the test data: -0.09597391258512444

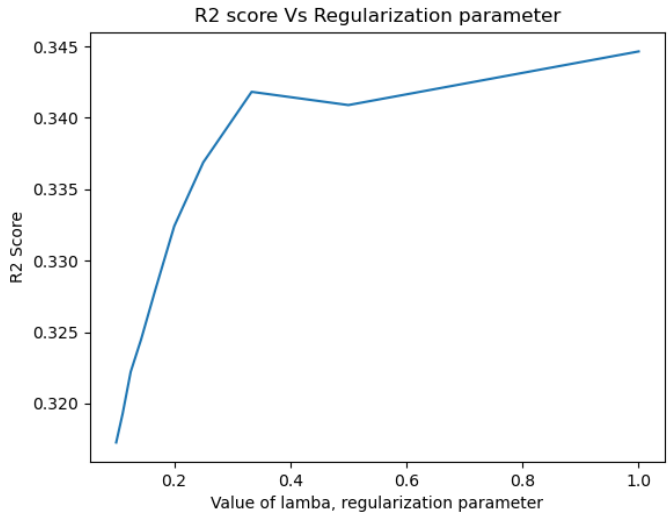




Kernel: poly
For Cmd_vel_v
Lambda - 1 to 10
R2 scores for training list (varying lambda) - [0.34465222638286896, 0.3408967707224049, 0.34182413476115703, 0.3368840547604155, 0.3324136199130906, 0.3278530559322499, 0.3244732430376348, 0.32221630096460896, 0.3192809202289748, 0.31726588250360577]

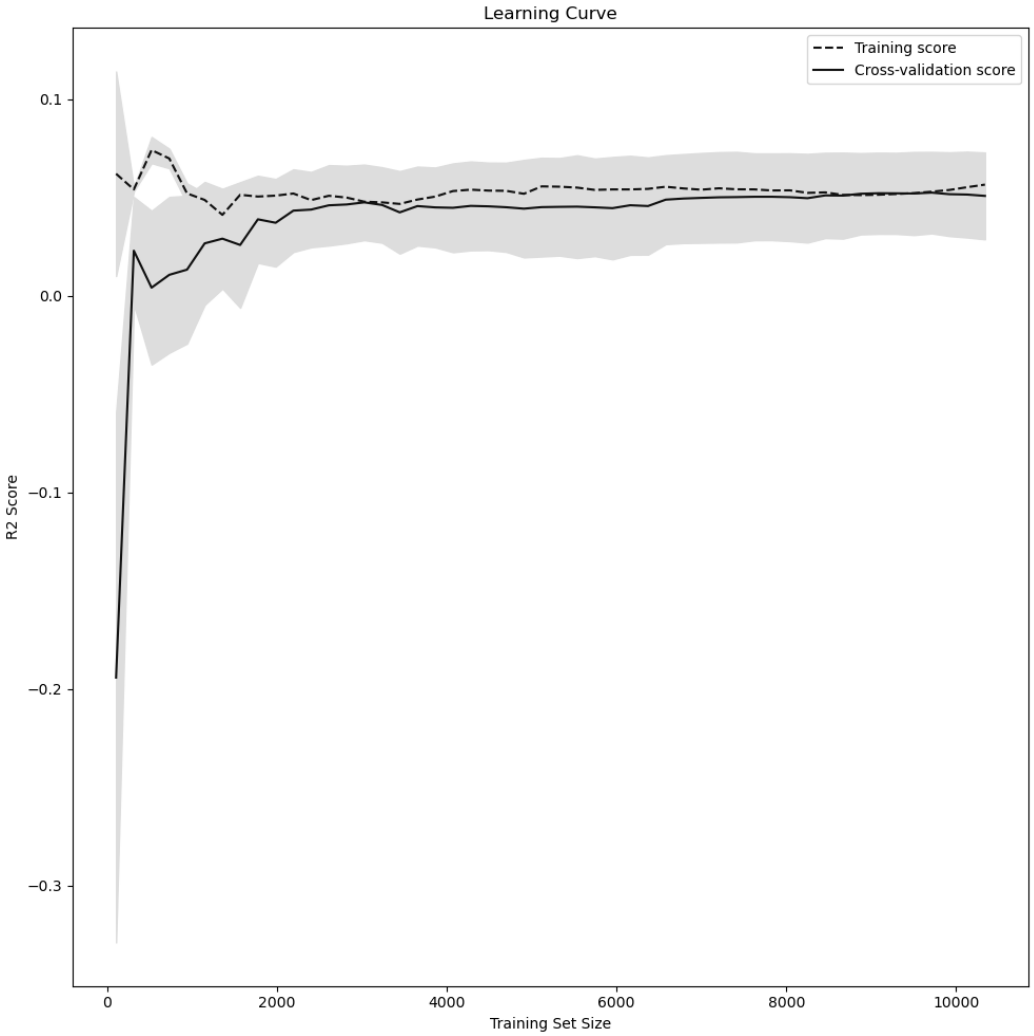
R2 score for the test data: 0.4233913792092908

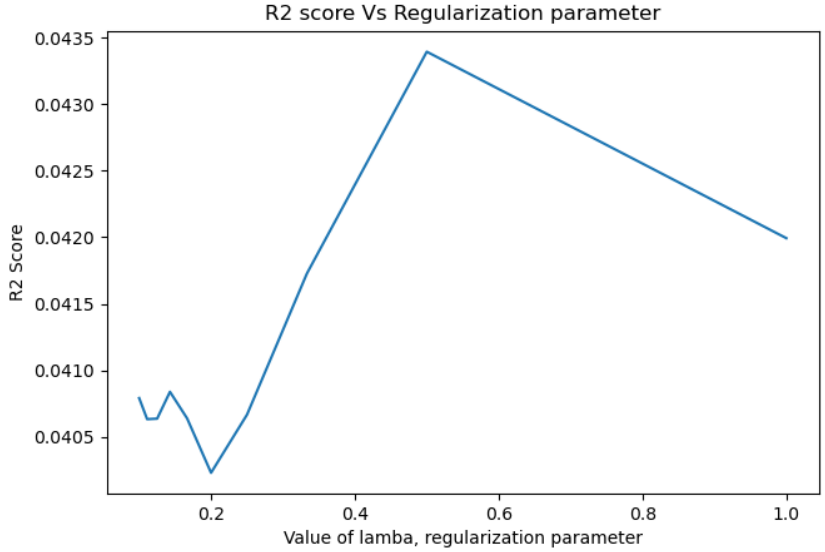




Kernel: poly
For Cmd_vel_w
Lambda - 1 to 10
R2 scores for training list (varying lambda) - [0.041992959789495576, 0.04339308429576261, 0.04172792801291081, 0.04066776552177942, 0.04023120384366141, 0.040639912085787544, 0.040839292761123636, 0.04063770614503015, 0.04063377737919882, 0.04079230915498144]

R2 score for the test data: -0.0884945064480751





XGBoost

XGBoost stands for Extreme Gradient Boosting. It uses more accurate approximations to find the best tree model. I am changing the max-depth hyper parameter. This is responsible for how deep each tree will grow during any boosting round. For each value of max-depth, I am calculating the R2 score of the actual output and predicted output. This was done for validation. Then I am taking the max-depth corresponding to the highest R2 score and then using that max-depth to train my model (by merging the train and validation set). After this my model is ready and can be tested on any external dataset.

Why XGBoost?

- Parallel Processing: XGBoost utilizes the power of parallel processing and that is why it is much faster than GBM. It uses multiple CPU cores to execute the model.
- Handling Missing Values: XGBoost has an in-built capability to handle missing values. When XGBoost encounters a missing value at a node, it tries both the left and right hand split and learns the way leading to higher loss for each node. It then does the same when working on the testing data.
- Cross Validation: XGBoost allows users to run a cross-validation at each iteration of the boosting process and thus it is easy to get the exact optimum number of boosting iterations in a single run. This is unlike GBM where we have to run a grid-search and only a limited value can be tested.

Training files used for corridor:

- July22_1.csv
- July22_2.csv
- July22_4.csv

Test file used for corridor: July22_52.csv

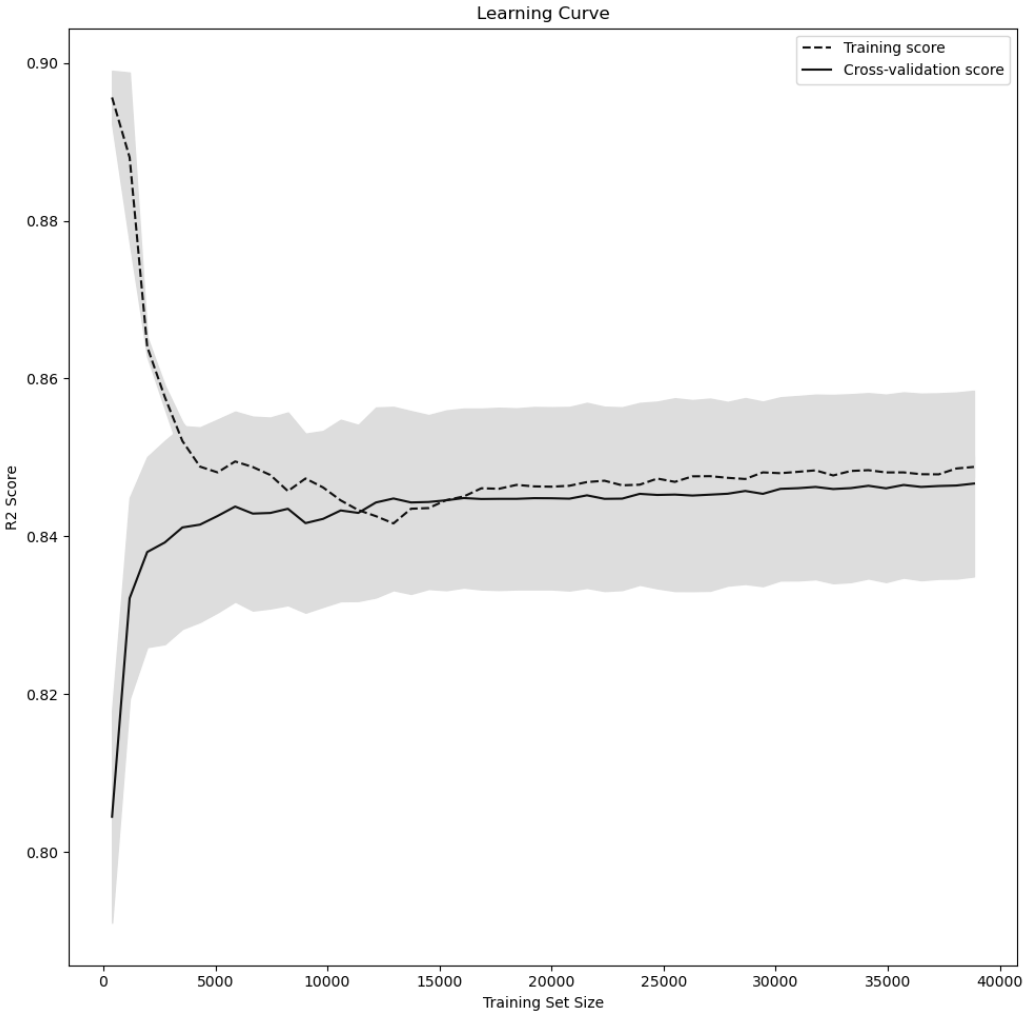
Training files used for open box:

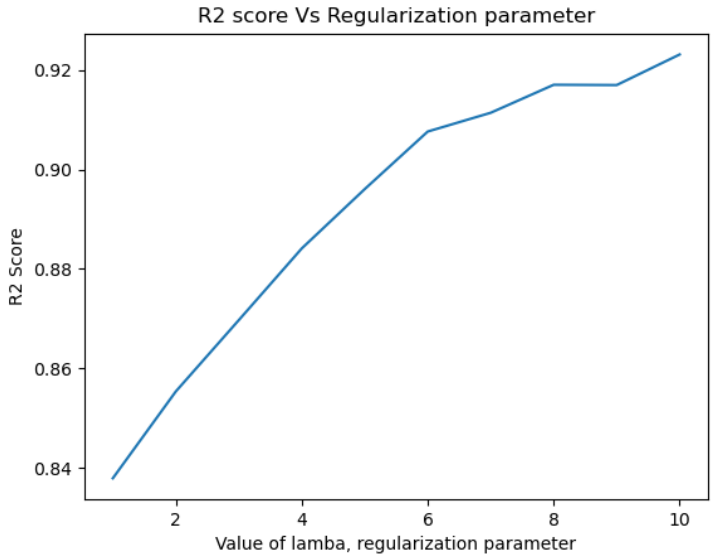
- Aug14_Box_1.csv
- Aug14_Box_2.csv
- Aug14_Box_4.csv

Test file used for open box: Aug14_Box_3.csv

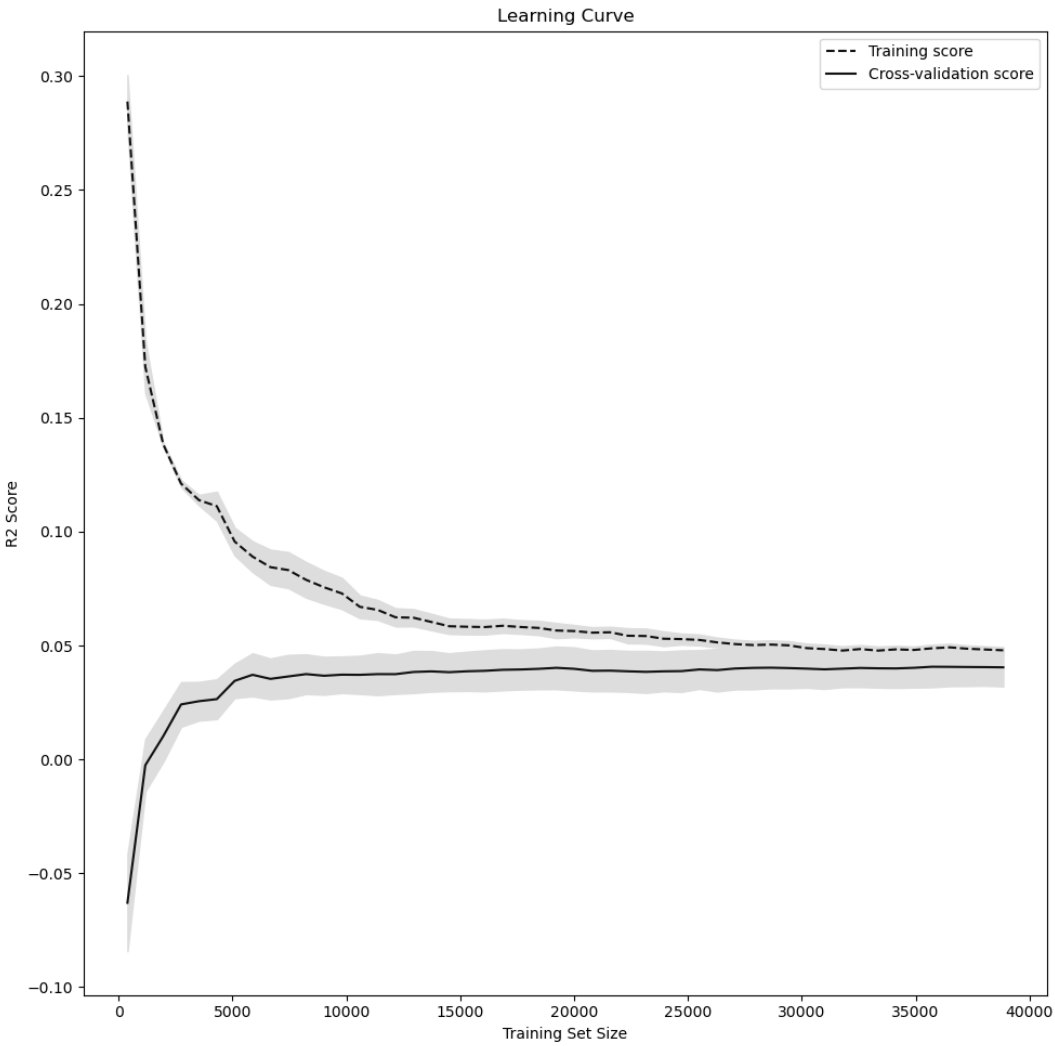
Data: OpenBox, max_depth = 10 for the learning curve
For cmd_vel_v
max_depth - 1 to 10
R2 scores for training list (varying max_depth) - [0.8378893614783738, 0.8553575853026602, 0.8697099360318804, 0.8841330914736946, 0.89607441901316, 0.9076505150429389, 0.9114279830182802, 0.9170605299401713, 0.917003170539876, 0.923155249728488]

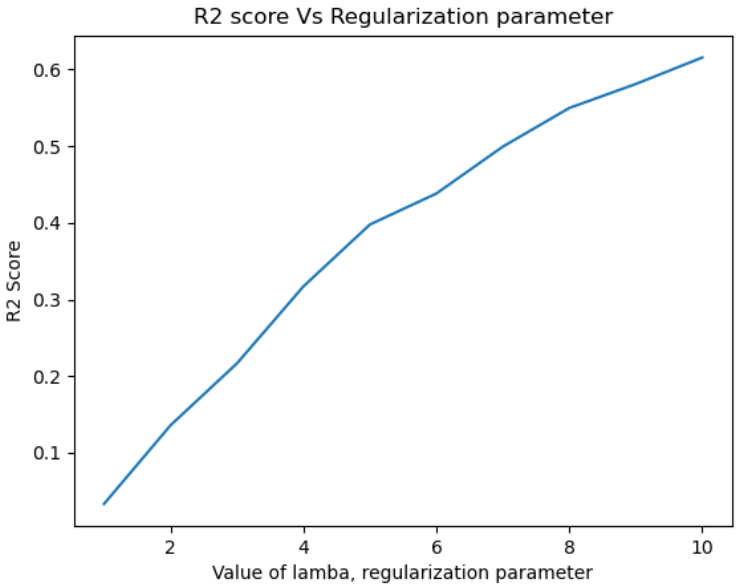
R2 score for the test data: 0.7565059895450211



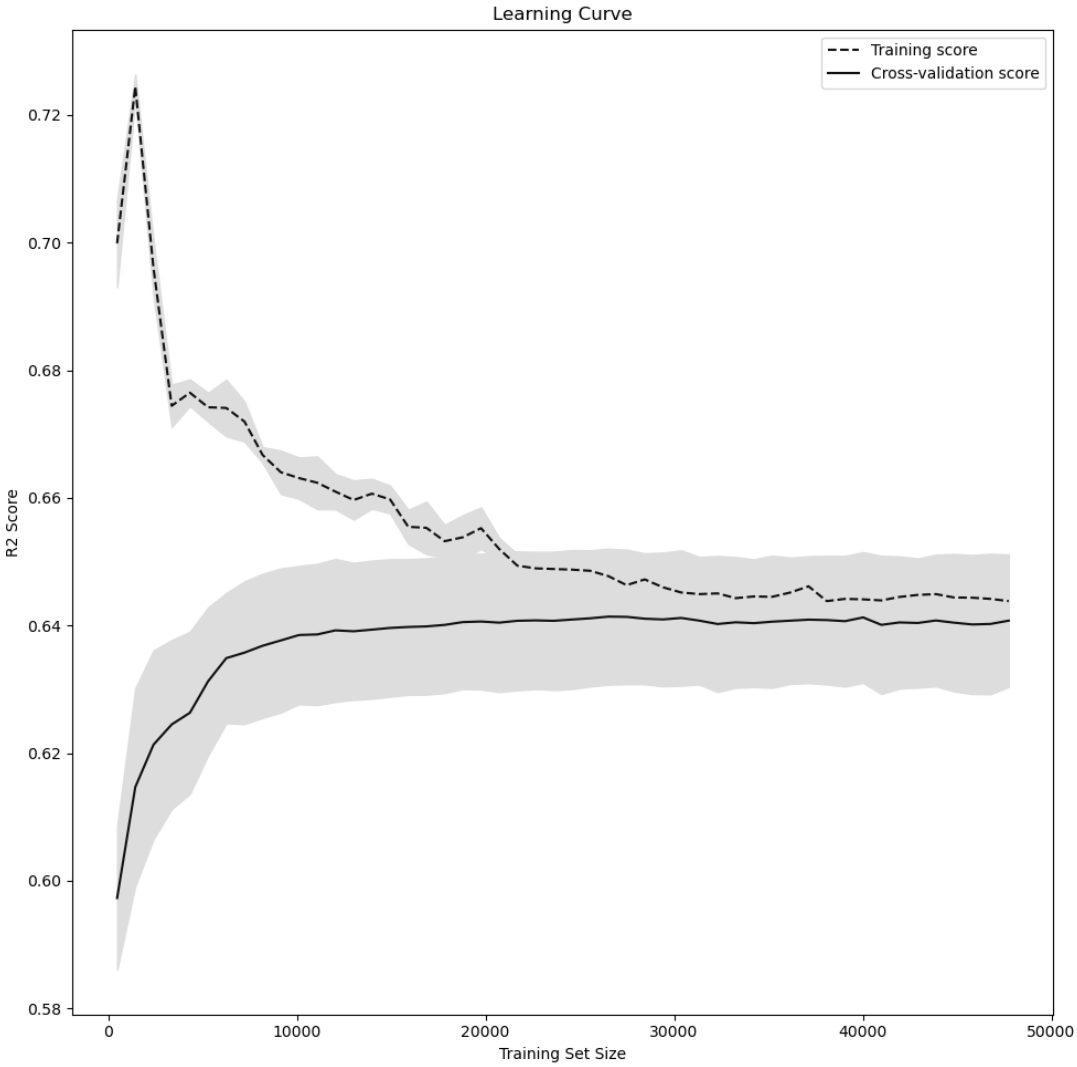


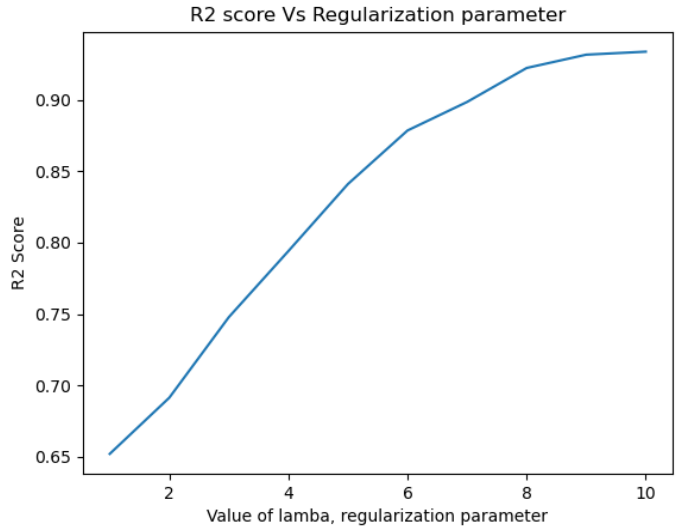
For cmd_vel_w
max_depth - 1 to 10
R2 scores for training list (varying max_depth) - [0.03342818293059824, 0.13590016748521427, 0.21669859120628876, 0.3171662255735812, 0.39780755233322007, 0.4382349813970656, 0.4994457491471572, 0.549849707856885, 0.5811078859824503, 0.615629557677176]
R2 score for the test data: -0.2594334449890192





Data: Corridor, max_depth = 10 for the learning curve
For cmd_vel_v
max_depth - 1 to 10
R2 scores for training list (varying max_depth) - [0.6520131300107418, 0.6913726515504606, 0.7478272949642766, 0.794107664520296, 0.8410743224462995, 0.878556154422747, 0.8985472885326838, 0.9223448696678516, 0.9316630566833949, 0.9337616741605381]
R2 score for the test data: 0.4131266087420681





For cmd_vel_w
max_depth - 1 to 10
R2 scores for training list (varying max_depth) - [0.05009120040267745, 0.15449273467240032, 0.25243563452869866, 0.3474057579072387, 0.4533375109490245, 0.5269760908422378, 0.6013593914575286, 0.6603086904011288, 0.7011135543941415, 0.7247936423963504]
R2 score for the test data: -0.6349991005113047