UNIVERSITY OF MARYLAND

ENPM808A FINAL PROJECT REPORT

- **By Rishabh Singh (M.Eng Robotics)**
- **117511208**

UNIVERSITY OF MARYLAND

General Project Pipeline followed

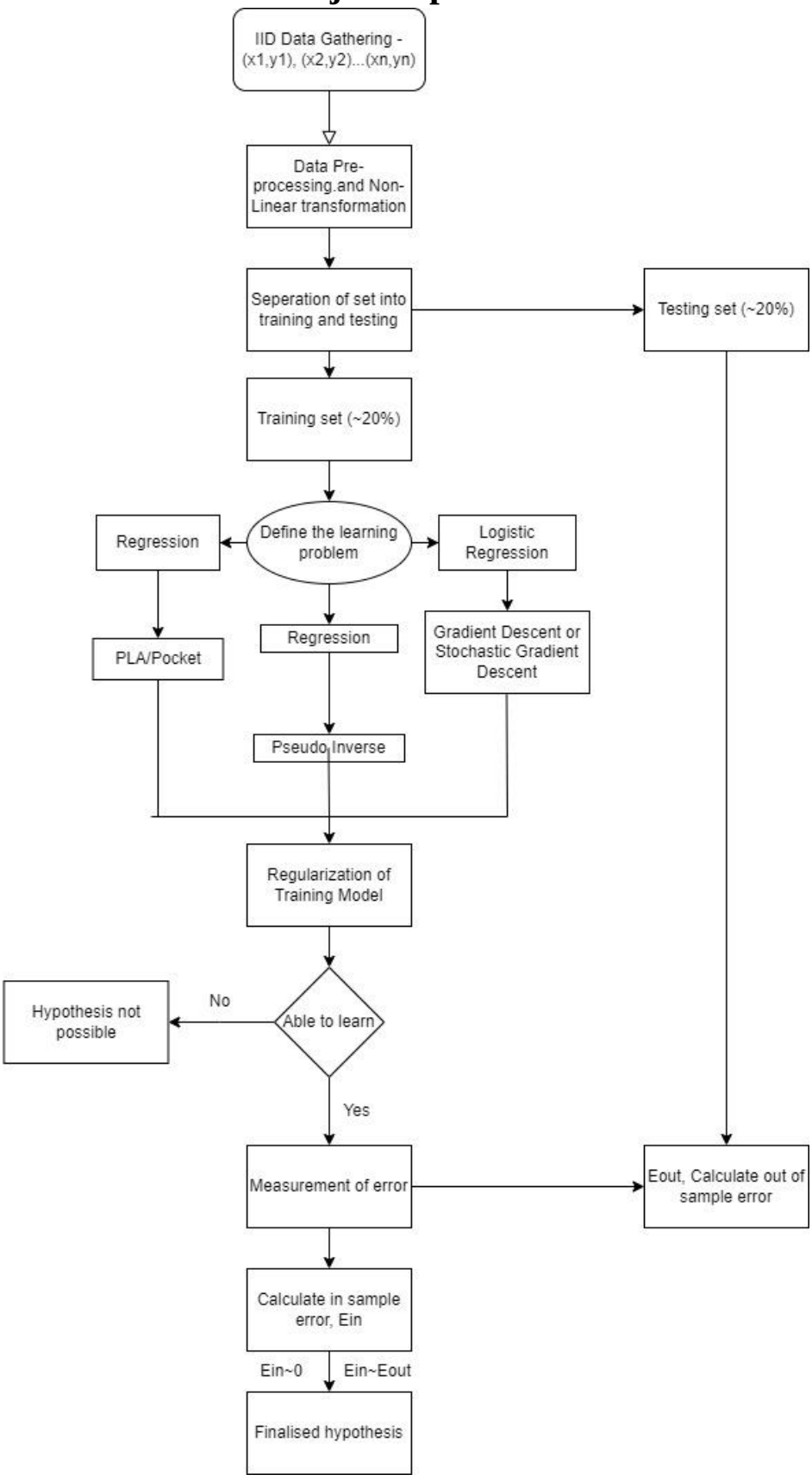

Fig 1. Pipeline

My codebase includes following classes implementation:

1) **Data_import.py –** Checks for csv in the input folder. Checks for any inconsistency. Combines the data and make a new csv to avoid loading multiple csv's again and again. Provides the data to the MPCLearning class present in pipeline.py module.
2) **Pipeline.**py – This module takes care of the major part of the pipeline
3) **SVM.py for support vector machine**
4) **NeuralNet.py for MLP**
5) **Regression.py for linear ridge regression**
6) **xg_boost.py for XG boost**
7) **utils.py for functions like quaternion to euler**

The steps shown above in the flowchart are described below:

Data Gathering

In this project, we were already given the data, and hence has been assumed that the data is IID but with two minimum scenarios:

1) A corridor scenario with moving obstacles Tested in a box scenario
2) An open box/hall environment with moving obstacles

Since the sampling rate, sampling time along with sampling space may differ hence I decided to train for both scenarios separately.

Data Selection

A large number of CSV files are provided. The simplest thing to do was **to take a random sub-sample with uniform distribution and check if it was significant or not**. If it's reasonably significant, we'll keep it. If it's not, we'll take another sample and repeat the procedure until we get a good significance level. Initially, I considered just 1 CSV file with more than enough data to split in the training and validation just to set my pipeline. Once set, I added enough files to gather enough data points to follow the VC dimension rule. Well, it is always good to have a large amount of data but I kept in mind two things:

1) Outliers
2) Computational complexity

The outliers were removed through the process of data cleaning. I created a data import module that takes all the CSV files in the folder, combines them, and write a new CSV. Writing a CSV is time-consuming, but it has to happen only in the first run, and is very convenient to use the generated CSV again and again for tuning parameters. The same thing is done for testing CSV files.

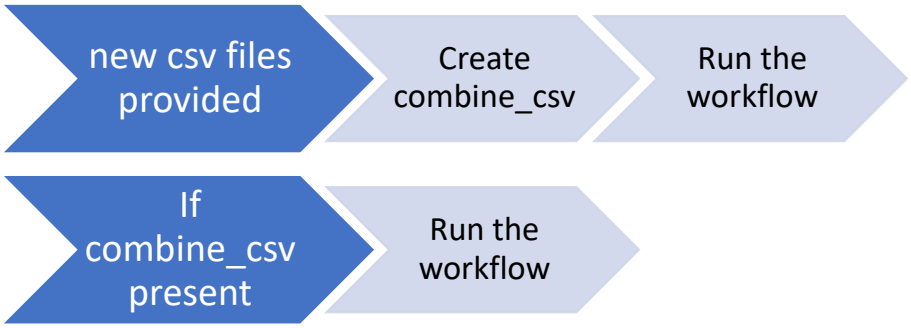

Fig 2. Data import steps

Data Cleaning and Pre-processing

This is performed to check following flaws with the input data:

1) Can be converted to NumPy float64 format for data type consistency?
2) Are the number of rows consistent?
3) Presence of any null value
4) Repeated/Stagnant or Unchanging data

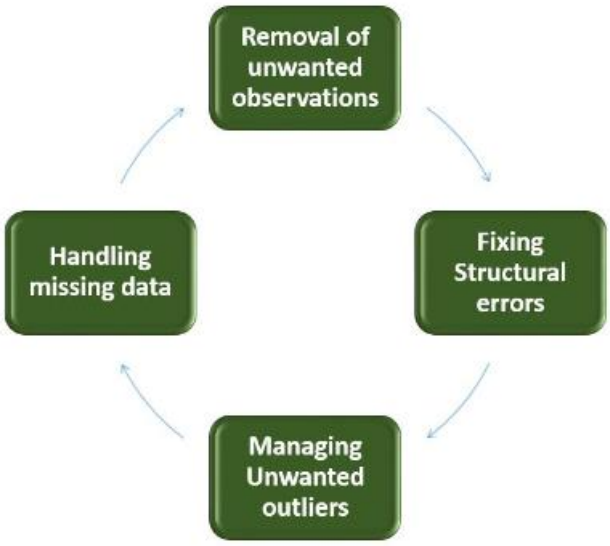

Fig 3. Data cleaning

Source: https://www.geeksforgeeks.org/

Part of this process is handled by the data_import.py and pipeline.py module provided in the code base. The rest is taken care by the pre-processing tool of sklearn library. As mentioned on their page, the **Standardization** of datasets is a **common requirement for many machine learning estimators** implemented in scikit-learn; they might behave badly if the individual features do not more or less look like standard normally distributed data: Gaussian with **zero mean and unit variance**, basically white.

Feature Selection

There are three main goals to feature selection.

1) Improve the accuracy with which the model can predict new data.
2) Reduce computational cost.
3) Produce a more interpretable model.

Through the given data I am trying to capture as much information as necessary for the controller The given data consists of Lidar data, robot position, local goal position, and final goal position. To define features and after knowing the scenario, I tried to think of both as a Controls engineer and as a Machine Learning engineer.

The 11 features defined are as follows:

- Distance from the local goal position is one of the costs in MPC.
- Distance from the final goal position is one of the costs in MPC.
- Angular error in shortest trajectory to local goal and vehicle heading angle is another cost in MPC

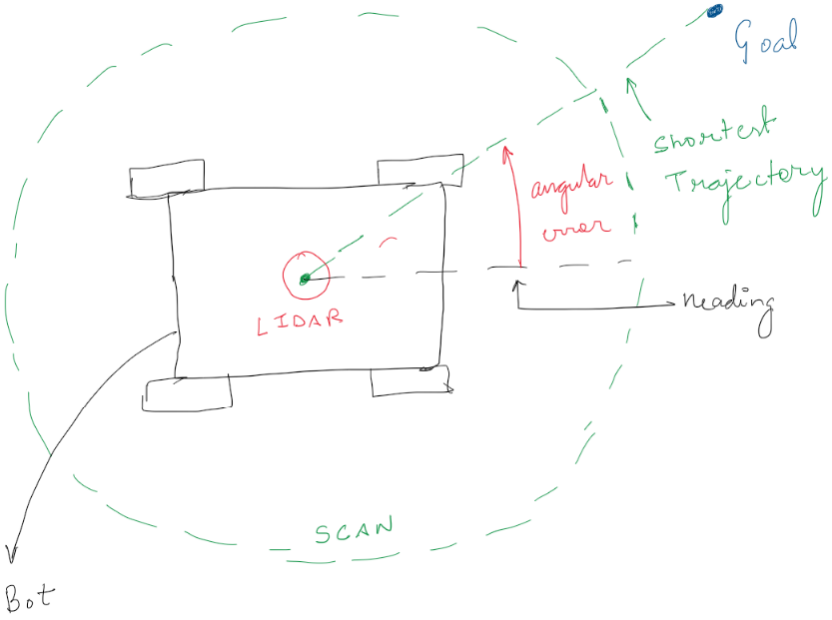

Fig 4. Angular error between bot's yaw and shortest trajectory

- Angular error in shortest trajectory to final goal and vehicle heading angle
- Lidar output in LOS (Line of Sight) of the robot is basically at the 540th index of lidar
- Lidar output towards the shortest trajectory for the local goal-
    - This was computed using the slope of the shortest trajectory and angular difference calculated above.
- Lidar output towards the shortest trajectory for the final goal
- Maximum distance the robot can travel towards the shortest trajectory for the local goal

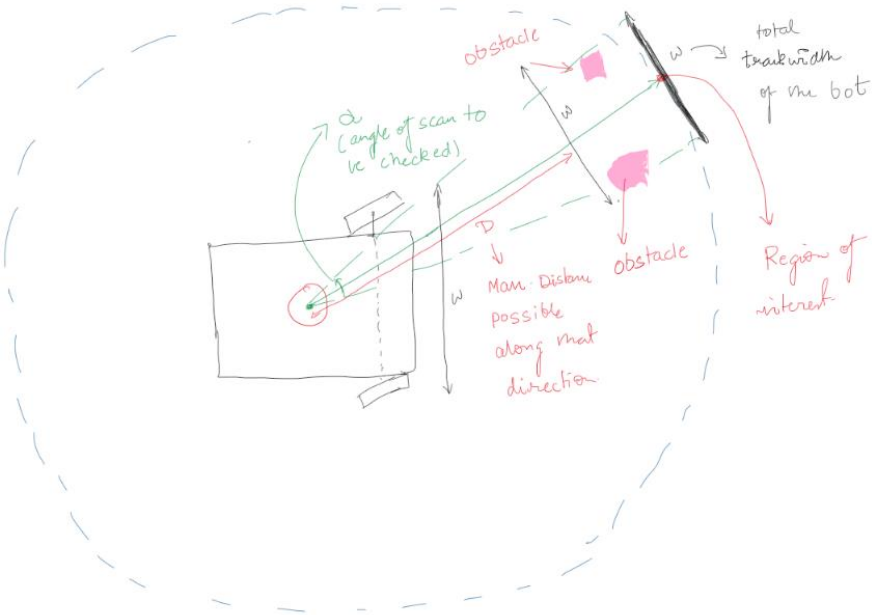

Fig. 5 Obstacle detection cost

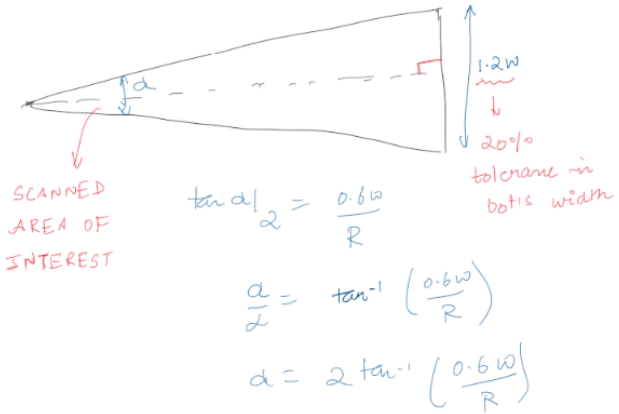

Fig 6. Obstacle detection with tolerance in width of the bot (assuming 30 cm of width)

- Maximum distance the robot can travel towards the shortest trajectory for the final goal
- Angular difference between bot's yaw and final pose
- Angular difference between bot's yaw and local pose

Note: The limitation in output velocity and omega is also be considered

- **Predicted velocity < Max bot velocity (found from given data)**
- **Predicted omega < Max bot omega (found from given data)**

There can be more such features, but these are the ones considered because of time limitations.

UNIVERSITY OF MARYLAND

Metrics Considered

These are defined inside the model classes for plotting and calculations. Since we care more about how accurate the predictions are w.r.t the actual values, we need the mean squared error and the variations spread to be low. Hence, the major metrics considered were:

- **R2 score**: It represents the proportion of variance (of y) that has been explained by the independent variables in the model. It provides an indication of goodness of fit and therefore a measure of how well-unseen samples are likely to be predicted by the model, through the proportion of explained variance.

  The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected (average) value of y, disregarding the input features, would get an R2 score of 0.0.

  If $\hat{y}_i$ is the predicted value of the i_{th} sample and y_i is the corresponding true value for total n samples, the estimated R^2 is defined as:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ and $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}\epsilon_i^2$.

SOURCE: https://scikit-learn.org

- **Mean Squared Error**: This needs to be as low as possible and helps in tuning hyperparameters.
  If $\hat{y}_i$ is the predicted value of the i_{th} sample and y_i is the corresponding true value for total n samples, the estimated MSE is defined as:

$$\mathrm{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2.$$

Model selection and test with box data:

The models considered for the projects are:

1) Linear Regression
2) SVM
3) XG Boost by Nvidia
4) Multilayer Perceptron

Common Pipeline considered for every model:

1) Get the features and data
2) Perform pre-processing
3) Split the input data into training and validation with an 80:20 ratio
4) Choose the model and hyperparameter (in this case one because of time complexity)
5) Train with the input and change the parameter to find the optimal weights
6) Merge validation data with training and find the new weights using the parameter value selected above
7) Predict the outputs from training and testing
8) Check in-sample, validation, and out-sample R2 scores and MSE.

Model 1: Linear Ridge Regression

This model solves a regression problem where the loss function is the linear least squares function and regularization is given by the l2-norm. Also known as Ridge Regression or Tikhonov regularization.
The reasons behind using this model are:

1) Medium Complexity as shown below

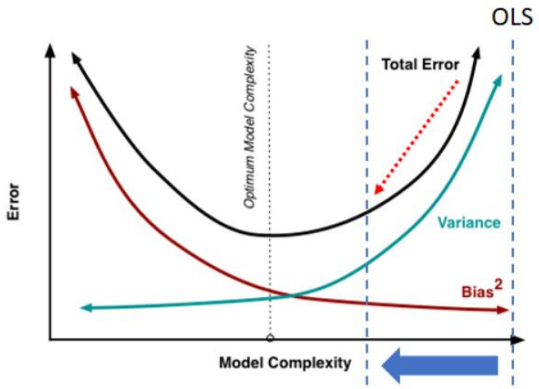

Fig. 7 Complexity Analysis

2) Regularization parameter considered allowing us to do hyperparameter tuning as desired.

• *Ridge regression* penalize the size of the regression coefficients based on their l^2 norm:

$$argmin_\beta \sum_i (y_i - \boldsymbol{\beta}' \boldsymbol{x}_i)^2 + \lambda \sum_{k=1}^{K} \beta_k^2$$

Fig. Regularization in regression

3) It protects the model from overfitting.
4) It does not need unbiased estimators
5) A Large amount of data is available and ridge regression helps in doing non-linear transformations and calculations with less complexity

Regularization and hyperparameter tuning:

To reduce the variance in the prediction and hence the out-sample error. The parameter chosen was lambda and was tuned for various values to optimize the output as will be shown in further sections.

Velocity Prediction (Input Data Size~50k points, Testing Data Size~15k points):

Parameter Tuning for R2 score and MSE:

- R2 Score variation with hyperparameter

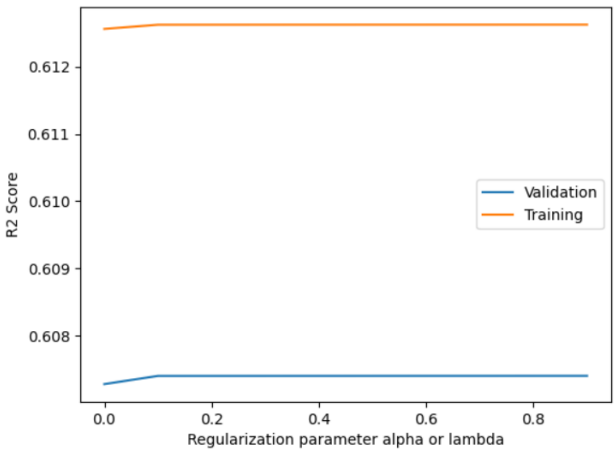

Fig. 8

In-sample R2 score = 0.612
Validation R2 score = 0.607
Out-sample R2 score = 0.602

- MSE variation with hyperparameter

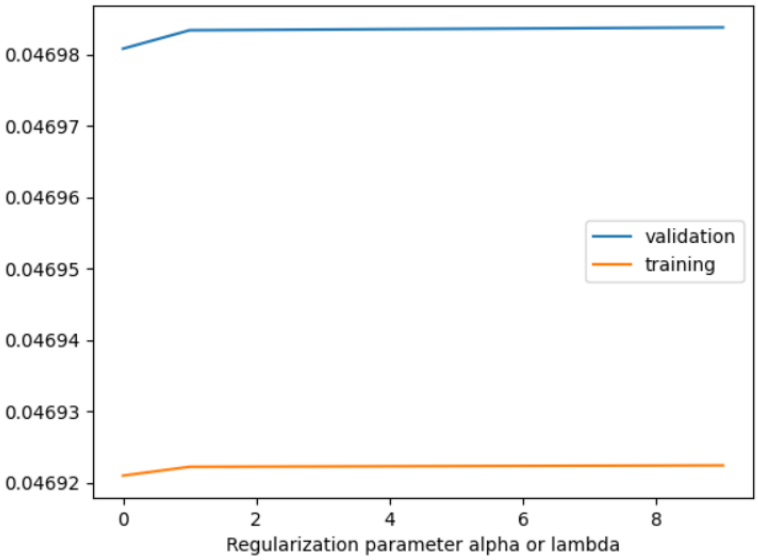

Fig. 9

In-sample MSE = 0.046920975145010944
Validation MSE = 0.04698079021276492
Out-sample MSE = 0.051465936370142046

- Learning and Validation Curve Velocity:

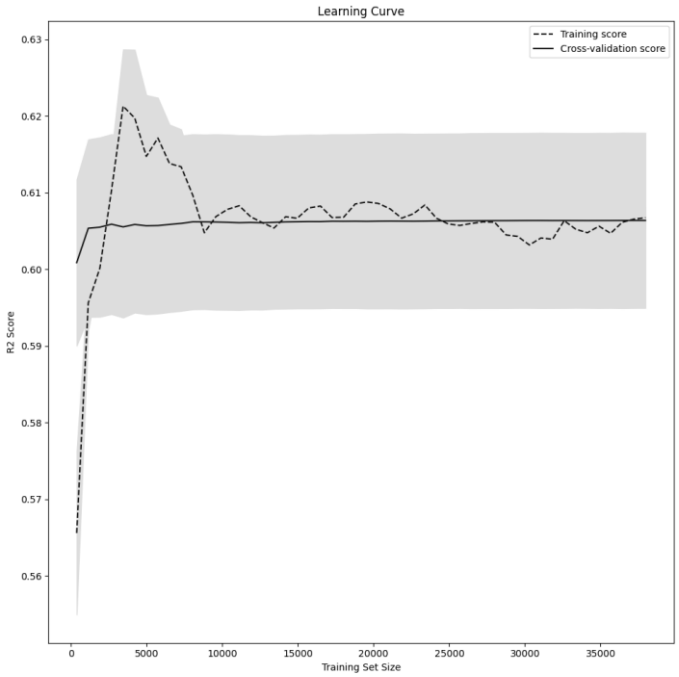

Fig. 10

Omega Prediction:

Parameter Tuning for R2 score and MSE:

- R2 Score variation with hyperparameter

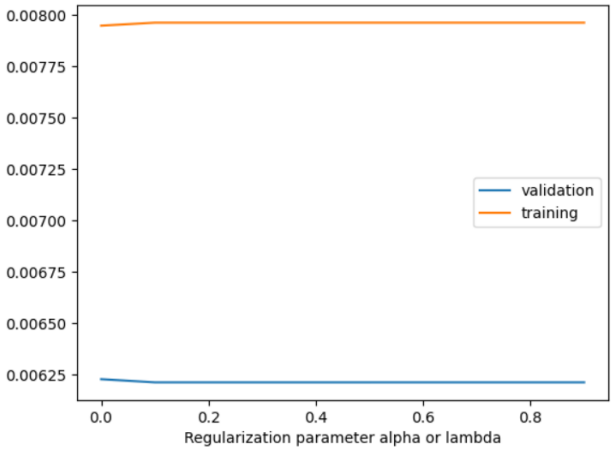

Fig. 11

In-sample R2 Score = 0.008

Validation R2 Score = 0.00625
Out-sample R2 Score = 0.0051

- MSE variation with hyperparameter

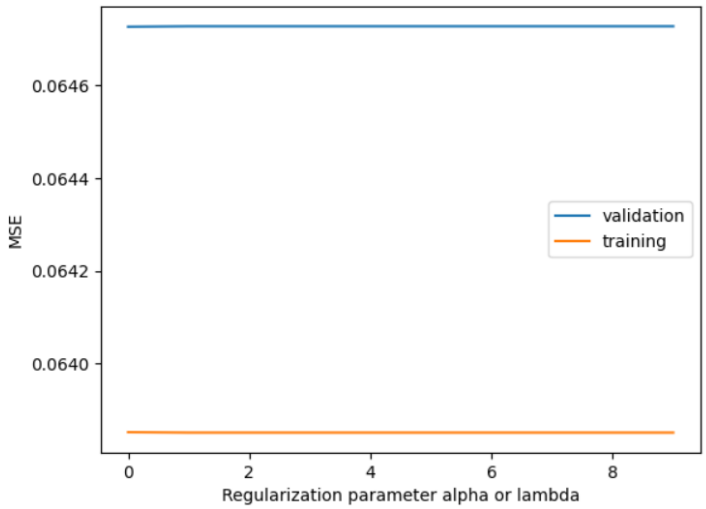

Fig. 12

In-sample MSE = 0.0638
Validation MSE = 0.0647
Out-sample MSE = 0.0628

- Learning and Validation Curve Omega:

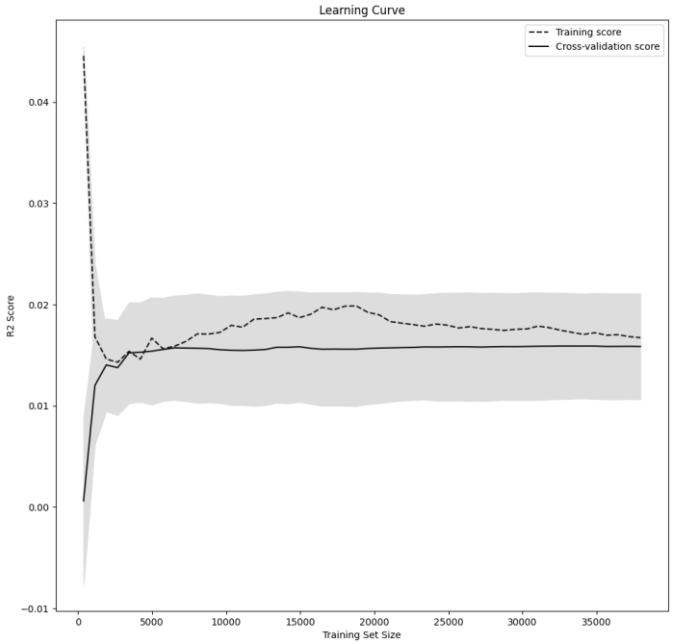

Fig. 13

Note: I merged more data into the training set and the performance output was improved somewhat

Velocity learning curve (Input Data Size~150k points):

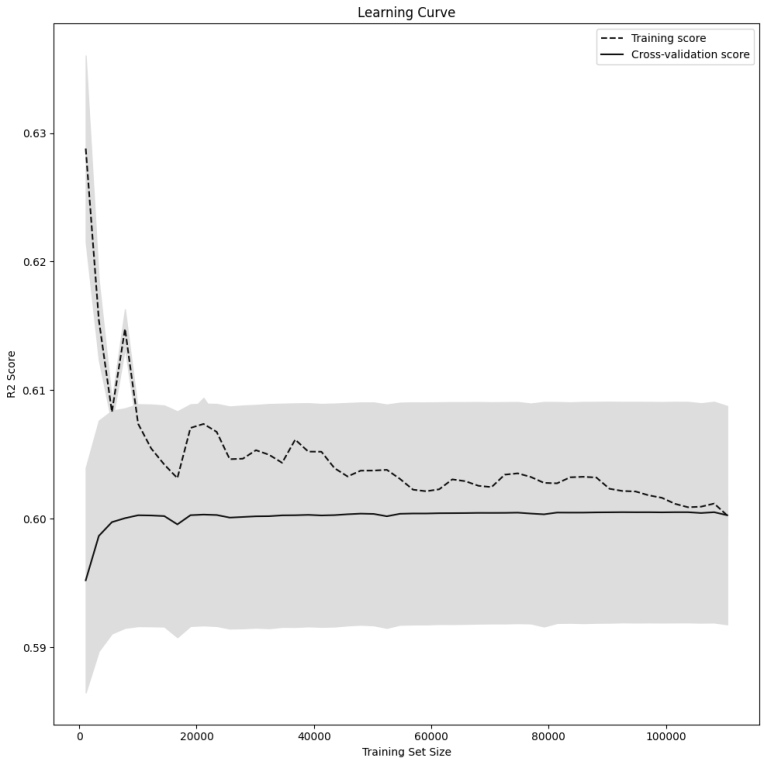

Fig. 14

Omega Learning Curve (Input Data Size~150k points):

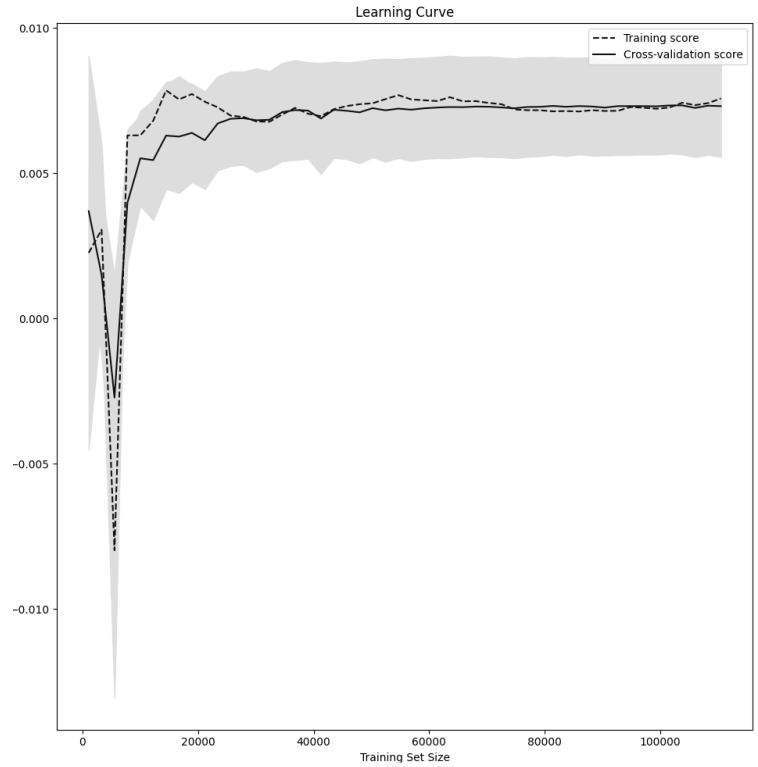

Fig. 15

UNIVERSITY OF MARYLAND

Model 2: SVM (SVR)

There is no specific reason I used this model. I just wanted to explore SVM as it provides a margin. This is not a good model for a large dataset but I tried to explore its capabilities with a reduced number of points. But it performed well for velocity predictions.

"The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. For large datasets consider using LinearSVC or SGDClassifier instead, possibly after a Nystroem transformer. Source: sklearn"

The major reasons I explored this are:

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

Regularization and hyperparameter tuning:

The parameter chosen was C/alpha. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty. and was tuned for various values to optimize the output as will be shown in further sections

I also tried to deploy different kernels and the results are as follows:

- **RBF Kernel:**

Velocity Prediction:

Parameter Tuning for R2 score and MSE:

- R2 Score variation with hyperparameter

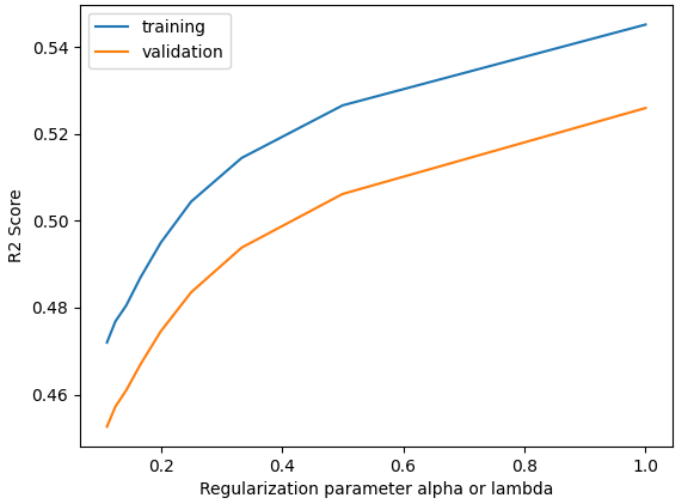

Fig. 16

In-sample R2 score = 0.58
Validation R2 score = 0.52
Out-sample R2 score = 0.48

- MSE variation with hyperparameter:

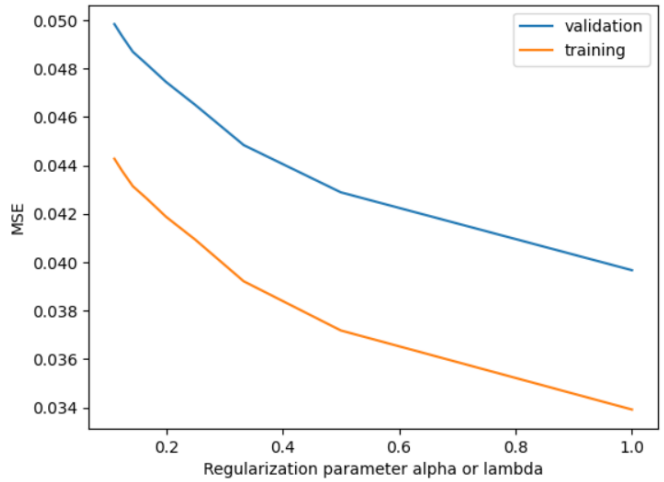

Fig. 17

In-sample MSE = 0.0443
Validation MSE = 0.045
Out-sample MSE = 0.048

- Learning and Validation Curve for Velocity:

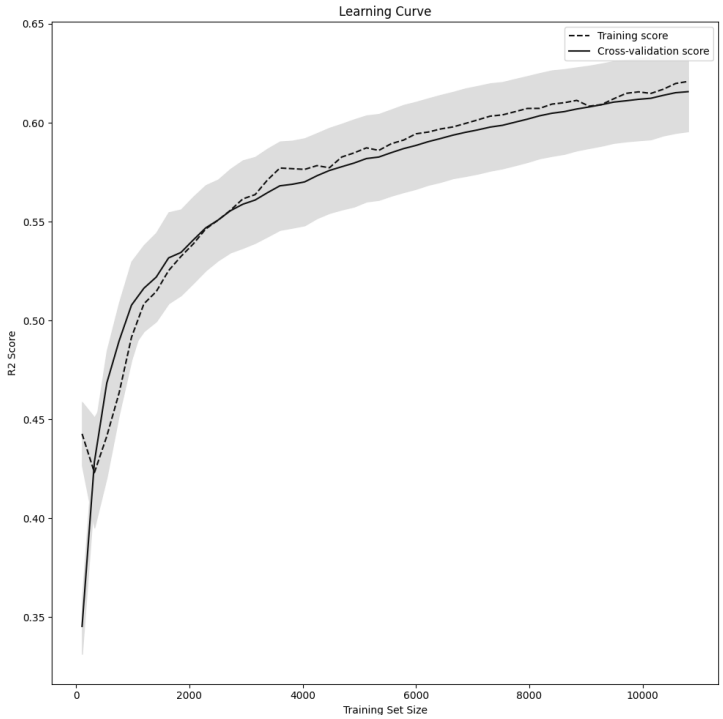

Fig. 18

Omega Prediction:

Parameter Tuning for R2 score and MSE:

- R2 Score variation with hyperparameter

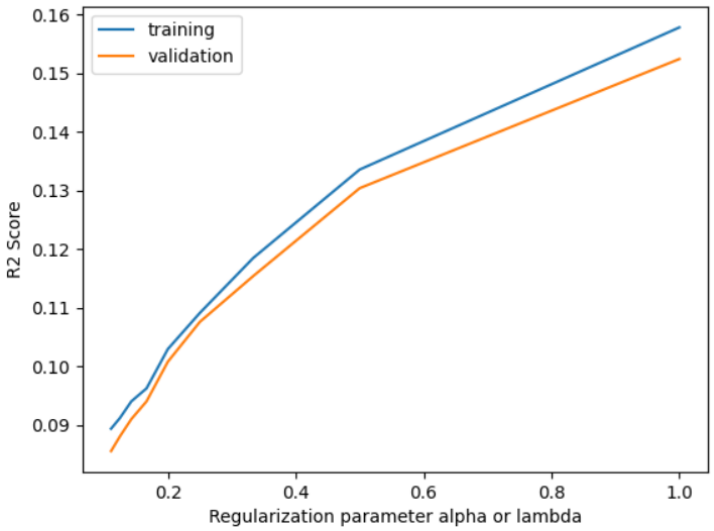

Fig. 19

In-sample R2 score = 0.16
Validation R2 score = 0.156
Out-sample R2 score = 0.14

- MSE variation with hyperparameter:

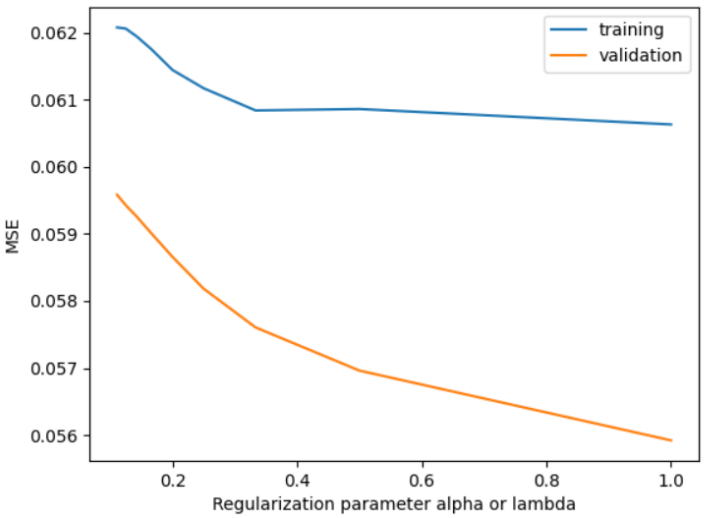

Fig. 20

In-sample MSE = 0.05958532990772832
Validation MSE = 0.062082018320186366
Out-sample MSE = 0.06019050373651059

- Learning and Validation Curve for Omega:

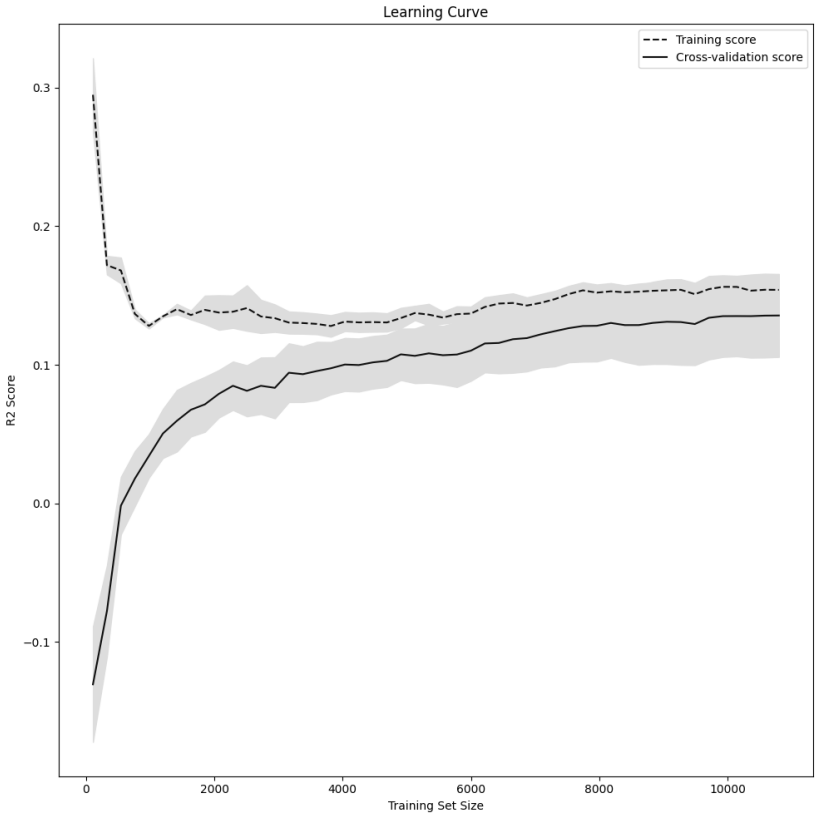

Fig. 21

- **Linear Kernel**: was not able to solve

- **Poly Kernel**: degree = 3

Velocity Prediction:

Parameter Tuning for R2 score and MSE:

- R2 Score variation with hyperparameter

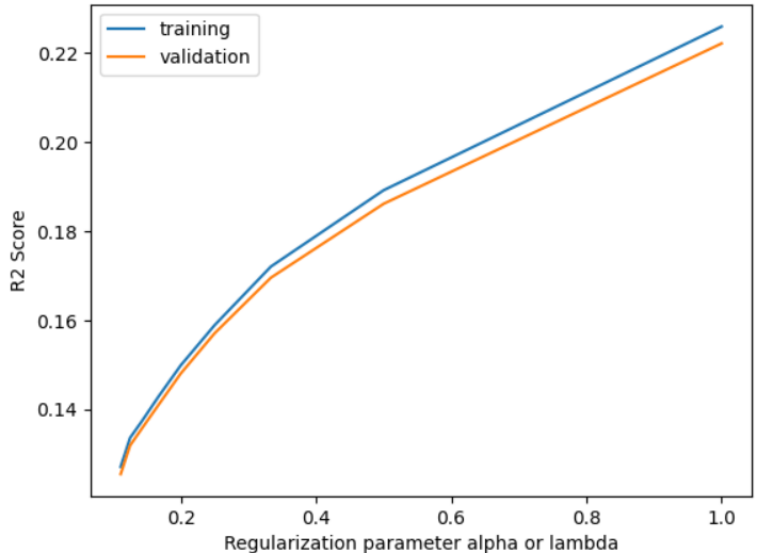

Fig. 22

In-sample R2 score = 0.23
Validation R2 score = 0.22
Out-sample R2 score = 0.20

- MSE variation with hyperparameter

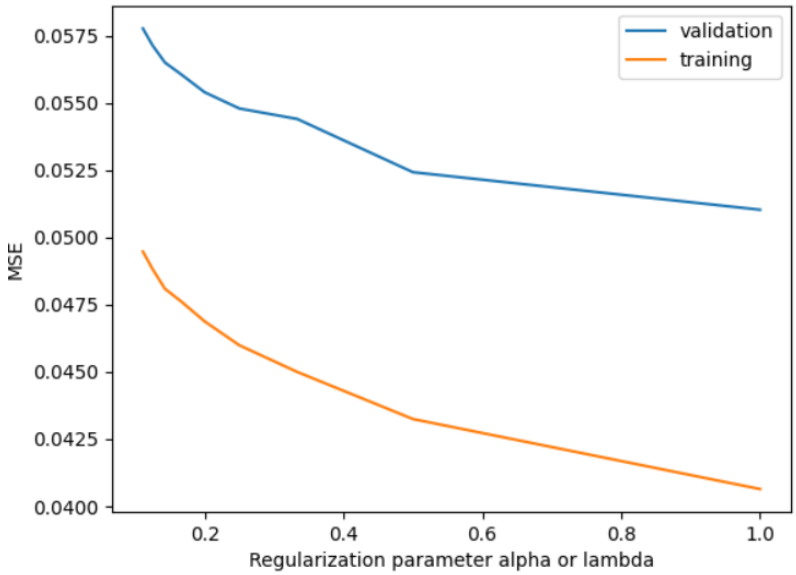

Fig. 23

- Learning and Validation Curve for Velocity:

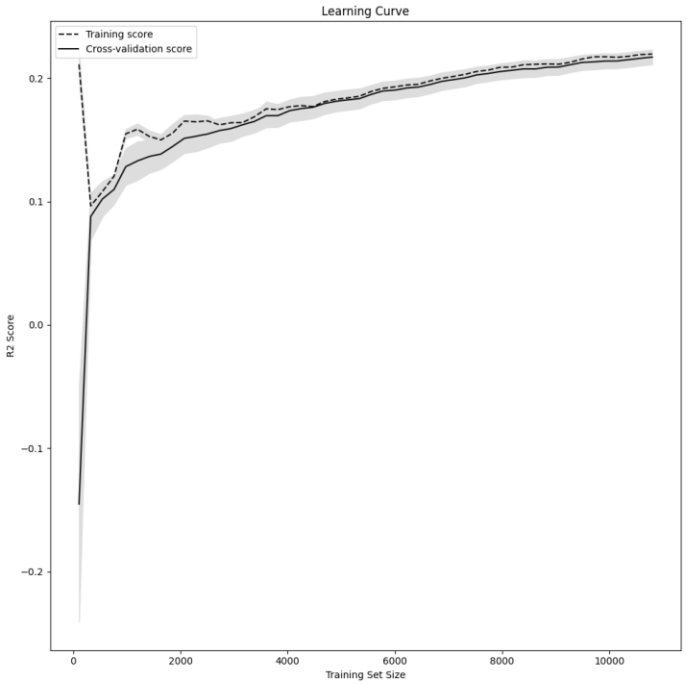

Fig. 24

Omega Prediction:

Parameter Tuning for R2 score and MSE:

- R2 Score variation with hyperparameter

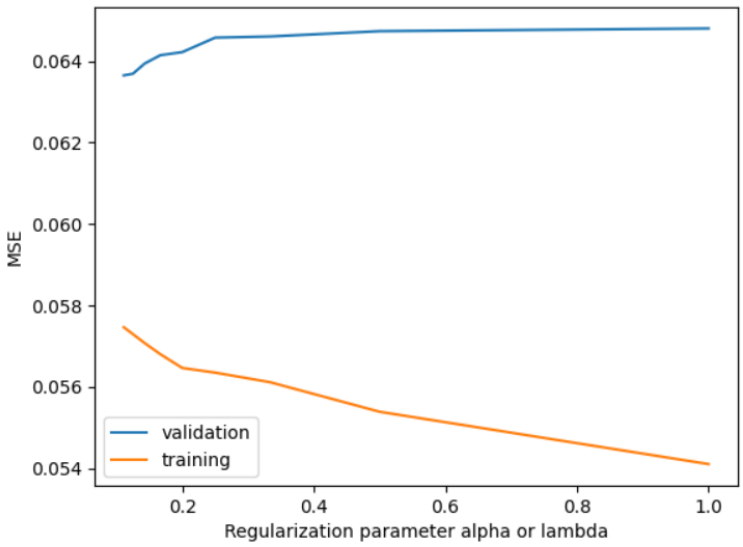

Fig. 25

In-sample R2 score = 0.54
Validation R2 score = 0.64
Out-sample R2 score = 0.13

- Learning and Validation Curve for Omega:

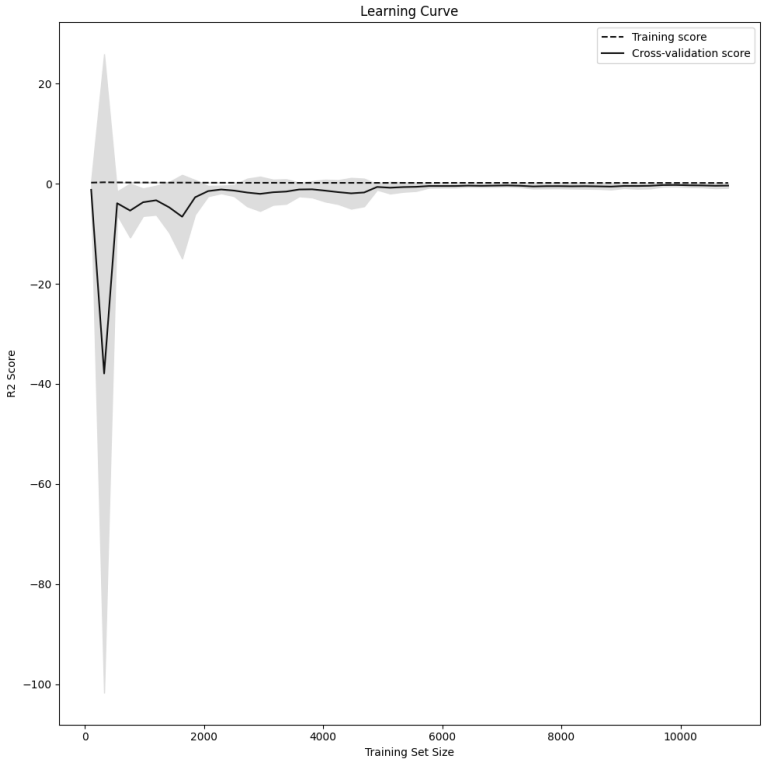

Fig. 26

- **Sigmoid Kernel:**

  The output from these kernels was negative R2 scores which meant that the data is fitting very badly with the predicted weights.

Model 3: XGBOOST (Source: NVIDIA.com):

The reason for using XG Boost is in its introduction itself. It really fast and optimized!!!!!

XGBoost is an open-source software library that implements optimized distributed gradient boosting machine learning algorithms under the **Gradient Boosting** framework.

XGBoost, which stands for Extreme Gradient Boosting, is a scalable, distributed **gradient-boosted** decision tree (GBDT) machine learning library. It provides parallel tree boosting and is the leading machine-learning library for regression, classification, and ranking problems. A Gradient Boosting Decision Trees (GBDT) is a decision tree **ensemble learning algorithm** similar to random forest, for classification and regression. Ensemble learning algorithms combine multiple machine learning algorithms to obtain a better model.

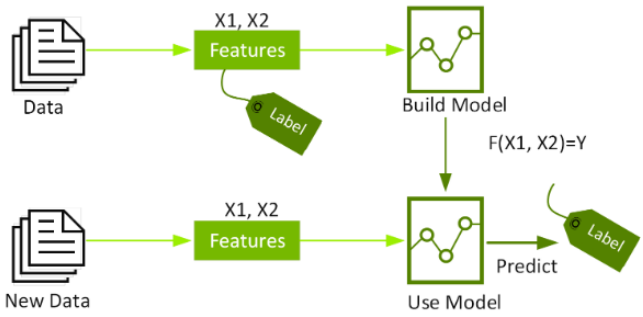

Fig. 27

Random forest uses a technique called bagging to build full decision trees in parallel from random bootstrap samples of the data set. The final prediction is an average of all of the decision tree predictions.
The term "gradient boosting" comes from the idea of "boosting" or improving a single weak model by combining it with a number of other weak models in order to generate a collectively strong model. **Gradient boosting** is an extension of boosting where the process of additively generating weak models is formalized as a gradient descent algorithm over an objective function. Gradient boosting sets targeted outcomes for the next model in an effort to minimize errors. Targeted outcomes for each case are based on the gradient of the error (hence the name gradient boosting) with respect to the prediction.

Regularization and hyperparameter tuning:

The parameter chosen was max_depth which is the Maximum depth in the tree search during the optimization of weights. Increasing this value might make the model more complex and more likely to overfit. This time I chose a different parameter to see how the model behaves. It was tuned for various values to optimize the output, as shown in further sections.

After tuning for max_depth, I took the best output (hopefully avoiding overfit) and then tuned for alpha.

Parameters tuned:

1) **Regularised alpha** – l2 norm regularization
2) **Max depth** - Maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit. 0 indicates no limit on depth. Beware that XGBoost aggressively consumes memory when training a deep tree
3) **Tree method** - Faster histogram optimized approximate greedy algorithm. GPU implementation of hist algorithm (gpu_hist)

Velocity Prediction:

Parameter Tuning for R2 score and MSE:

- R2 Score variation with hyperparameter
    - Max_depth

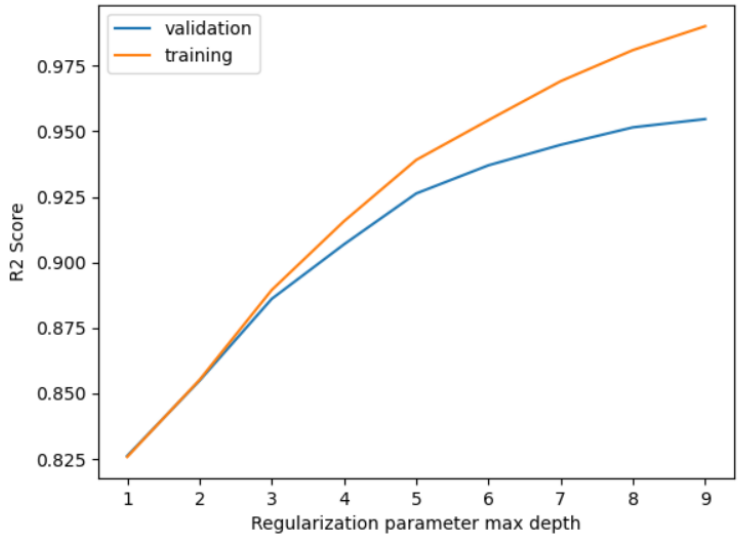

Fig. 28

In-sample R2 score = 0.962827 (one likely not overfitting)
Validation R2 score = 0.92244
Out-sample R2 score = 0.88950

- Lambda with max_depth = 9

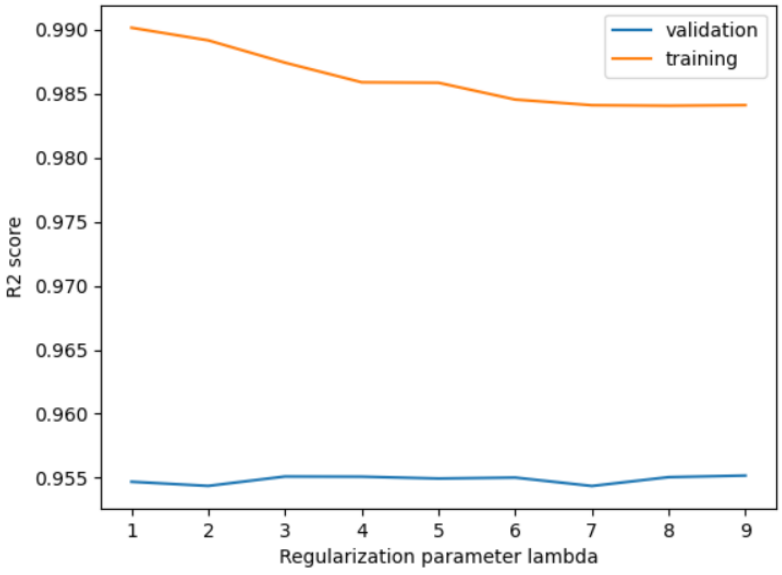

Fig. 29

Best performance is with lambda = 9

In-sample R2 score = 0.9840
Validation R2 score = 0.9551
Out-sample R2 score = 0.8784

- MSE variation with hyperparameter
  - Max_depth

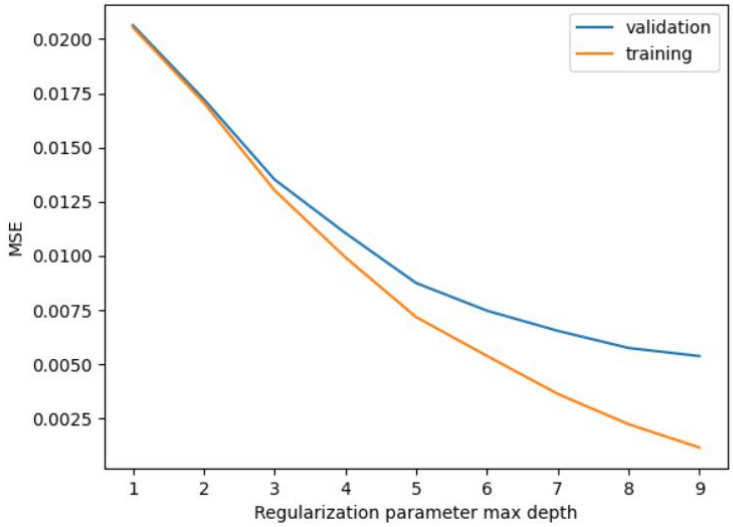

Fig. 30

In-sample MSE = 0.00199
Validation MSE = 0.0063
Out-sample MSE = 0.01331

  - Lambda with max_depth = 9

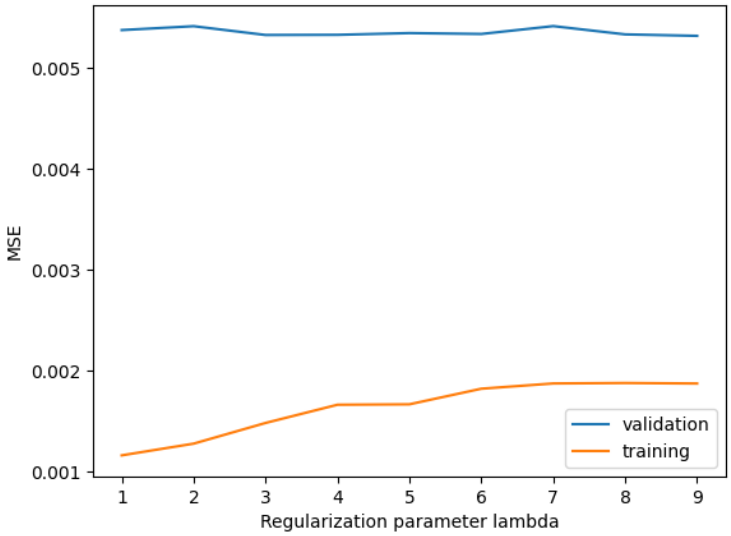

Fig. 31

In-sample MSE = 0.0018
Validation MSE = 0.0053
Out-sample MSE = 0.0146

Note: In sample and validation error decreased but out-sample increased by a bit at lambda = 9

- Learning and validation curve for velocity:

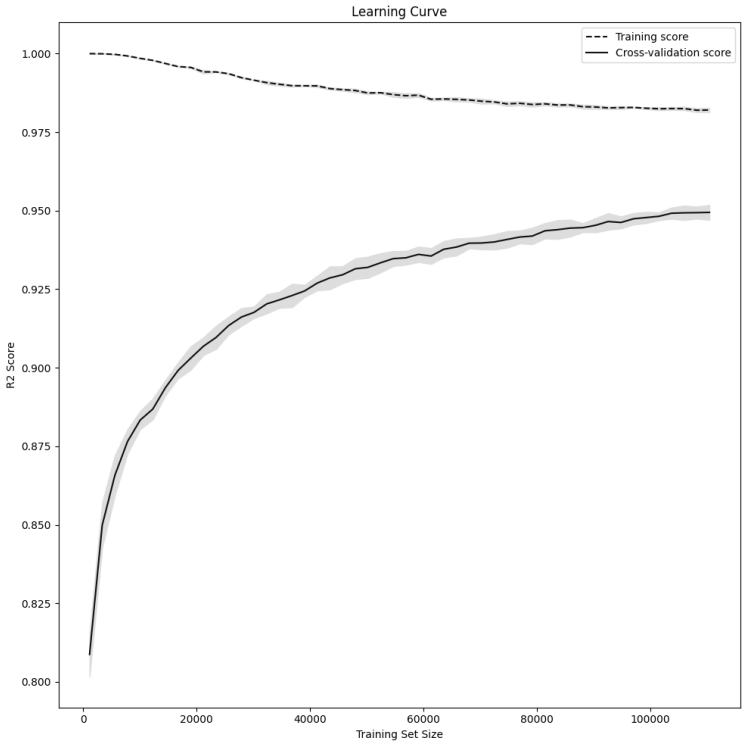

Fig. 32

Omega Prediction:

Parameter Tuning for R2 score and MSE:

- R2 Score variation with hyperparameter
  - Max_depth

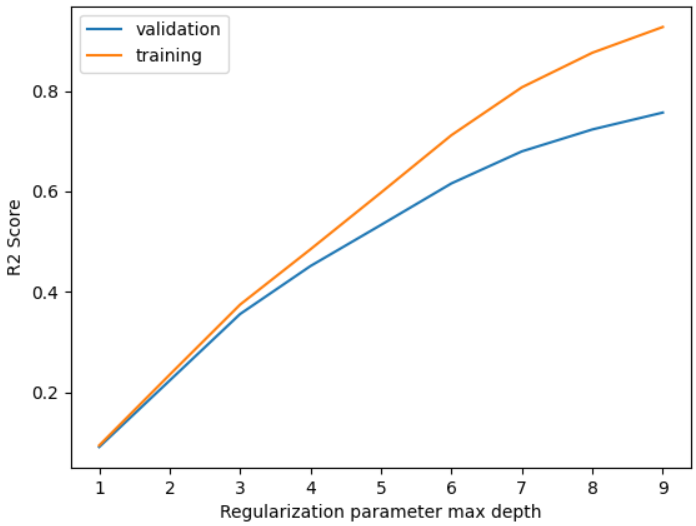

Fig. 33

In-sample R2 score = 0.8807
Validation R2 score = 0.69446
Out-sample R2 score = 0.3016

- Lambda with max_depth = 9

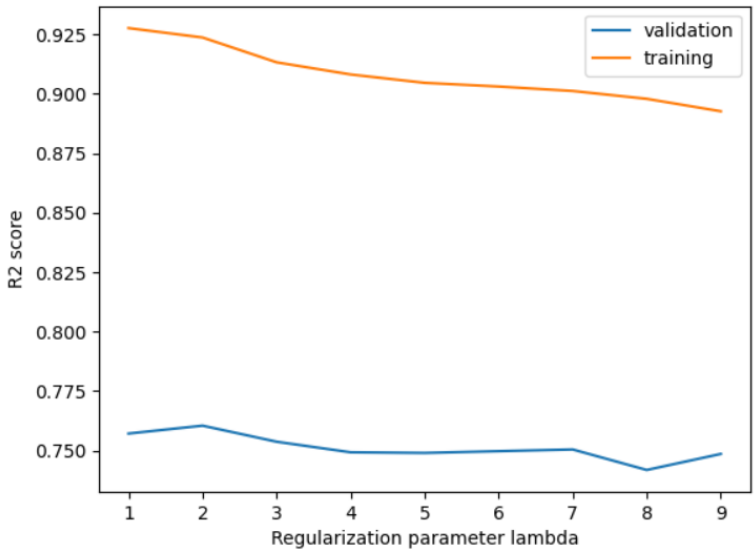

Fig. 34

The score came out best with lambda = 2
In-sample R2 score = 0.9237
Validation R2 score = 0.7605
Out-sample R2 score = 0.31185

- MSE variation with hyperparameter
  - Max_depth

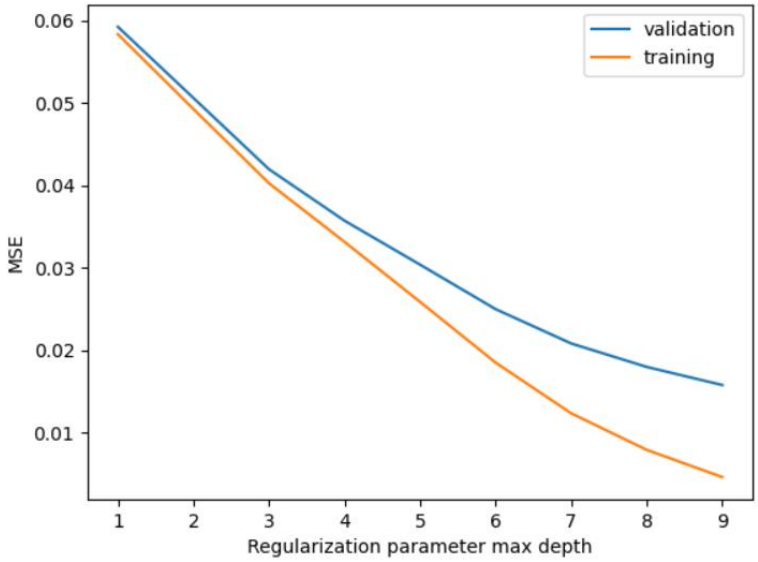

Fig. 35

In-sample MSE = 0.007678
Validation MSE = 0.019899
Out-sample MSE = 0.04383

  - Alpha tuning with max_depth = 8

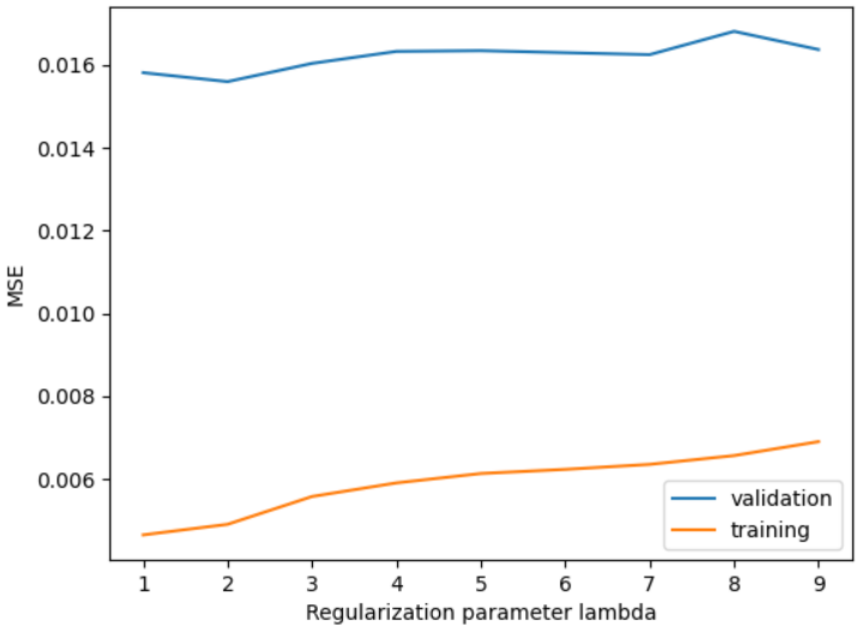

Fig. 36

In-sample MSE = 0.0049
Validation MSE = 0.01559
Out-sample MSE = 0.04319

Note: All the errors decreased at lambda = 2

- Learning and validation curve for Omega:

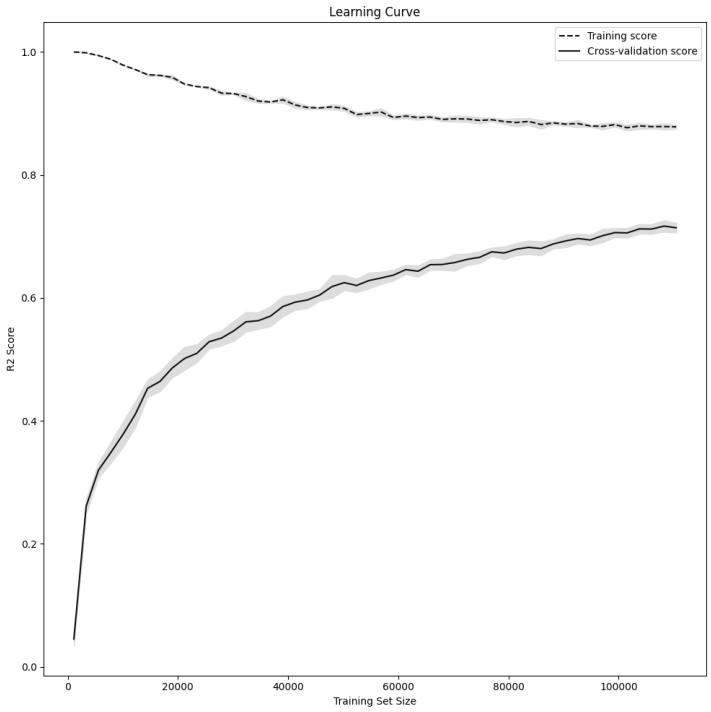

Fig. 37

After increasing data points:
In-sample R2 score = 0.8807
Validation R2 score = 0.69446
Out-sample R2 score = 0.3016

Note: The linear model deprecated in favor of squared error due to large noise in data set

Model 4: NEURAL NETWORK

Multi-layer Perceptron (MLP) (Source: scikit-learn.org) is a supervised learning algorithm that learns a function by training on a dataset, where m is the number of dimensions for input and o is the number of dimensions for output. It is different from logistic regression, in that between the input and the output layer, there can be one or more non-linear layers, called hidden layers. The figure below shows a hidden layer.

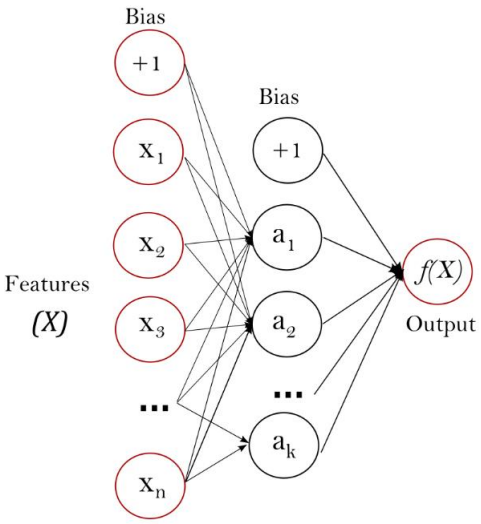

Fig. 38: Multilayer perceptron model

The reason I chose this?

1) Availability of large data set
2) It can be used to solve complex nonlinear problems.
3) Makes quick predictions after training.
4) The same accuracy ratio can be achieved even with smaller samples.

Also, I wanted to explore a Neural Network model. I used (10,2) as the size of hidden layer.

Velocity Prediction:

Parameter Tuning for R2 score and MSE:

- R2 Score variation with hyperparameter

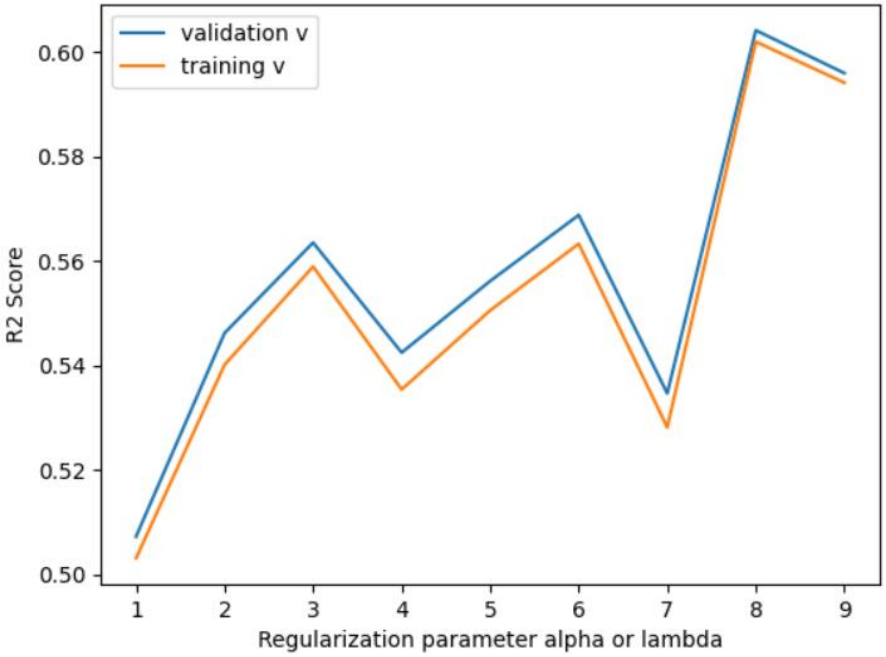

Fig. 39

In-sample R2 score = 0.610
Validation R2 score = 0.615
Out-sample R2 score = 0.591

- MSE variation with hyperparameter

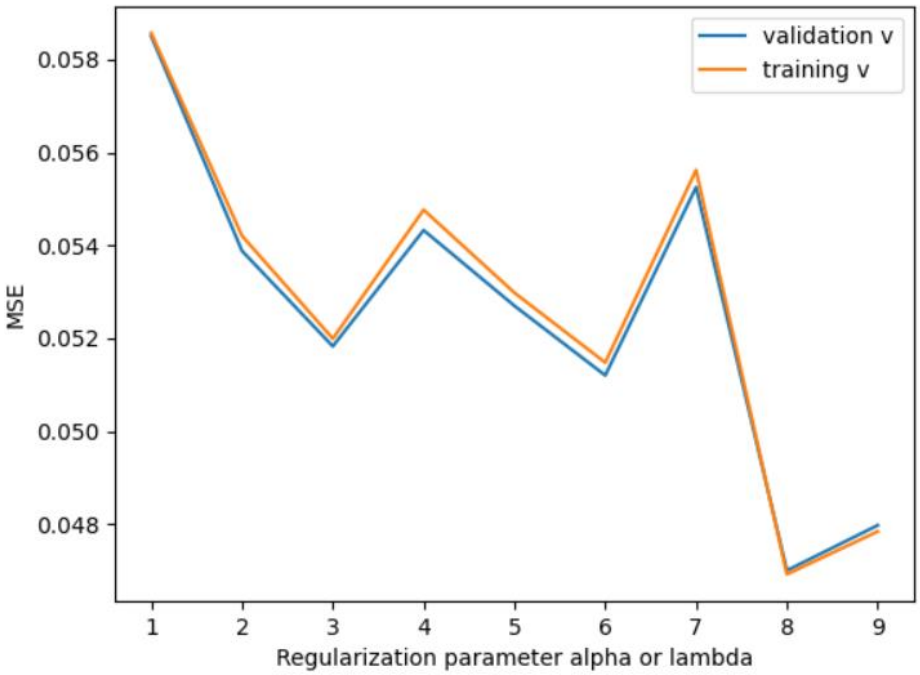

Fig. 40

In-sample MSE = 0.04691647128188403
Validation R2 score = 0.04699860370372856

Out-sample R2 score = 0.019885021045387995

- Learning and Validation Curve for velocity:

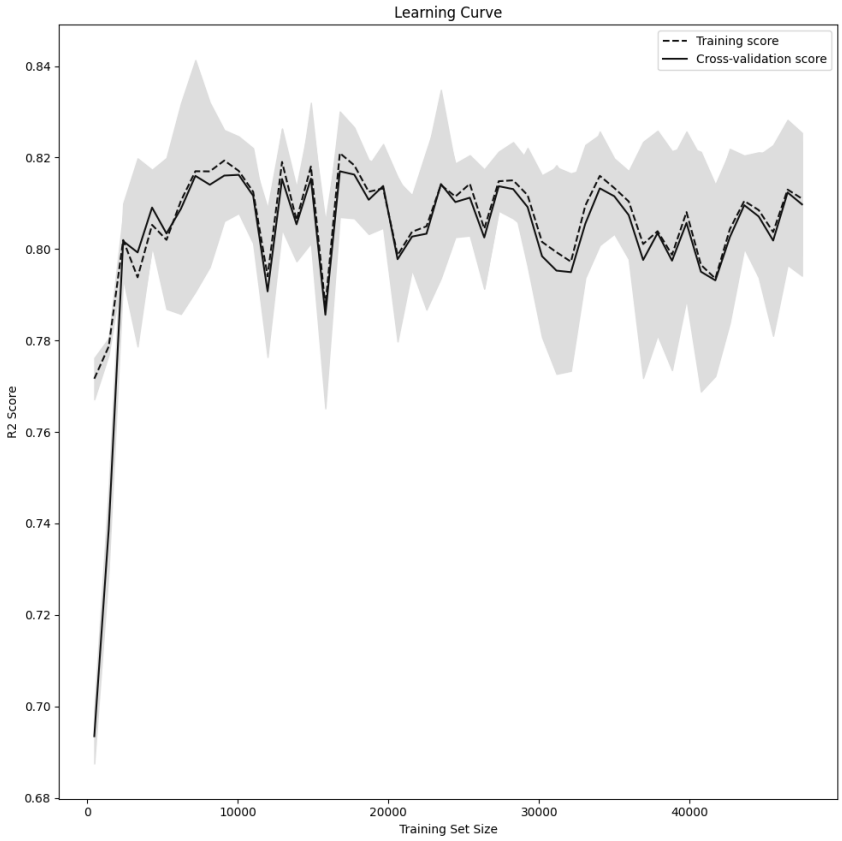

Fig. 41

Omega Prediction:

Parameter Tuning for R2 score and MSE:

- R2 Score variation with hyperparameter

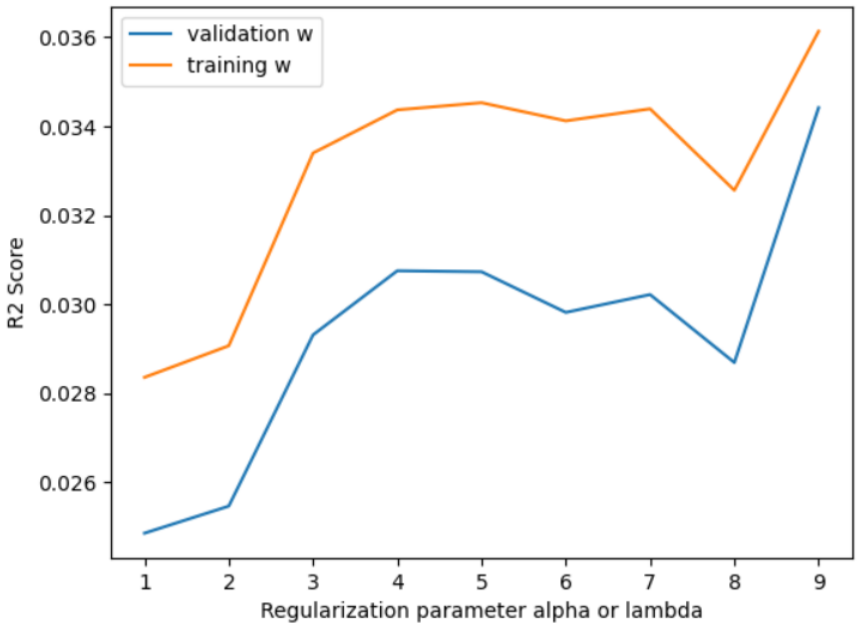

Fig. 42

In-sample R2 score = 0.038
Validation R2 score = 0.036
Out-sample R2 score = 0.03

- MSE variation with hyperparameter

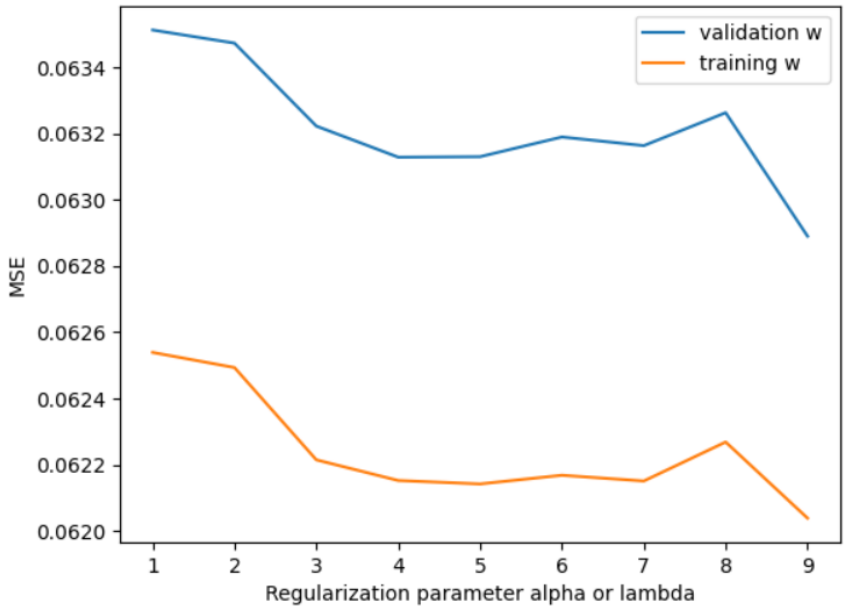

Fig. 43

In-sample MSE = 0.06203
Validation MSE = 0.06289
Out-sample MSE = 0.0695

- Learning and Validation Curve Omega

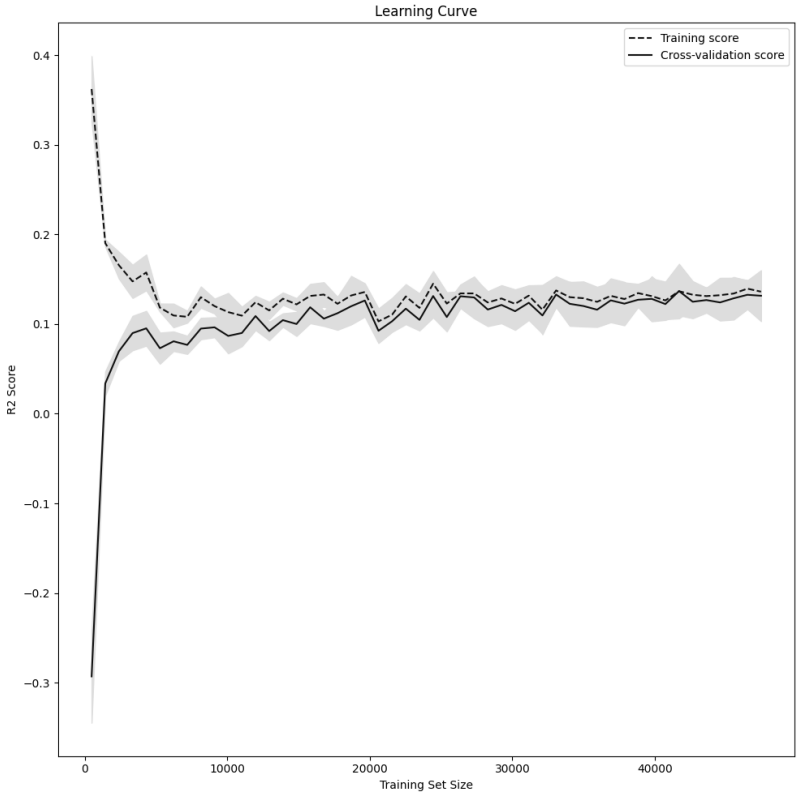

Fig. 44

Delivered Model:

With all the analysis shown above, the model I would like to deliver is **XG Boost** the reason is its fast speed and accuracy. Though it seems like an overfit but to make sure I will again verify the learning pipeline with the corridor data in hand.

UNIVERSITY OF MARYLAND

Dataset 2

Model test with corridor data:

Model 1: Linear Ridge Regression:

Velocity MSE:

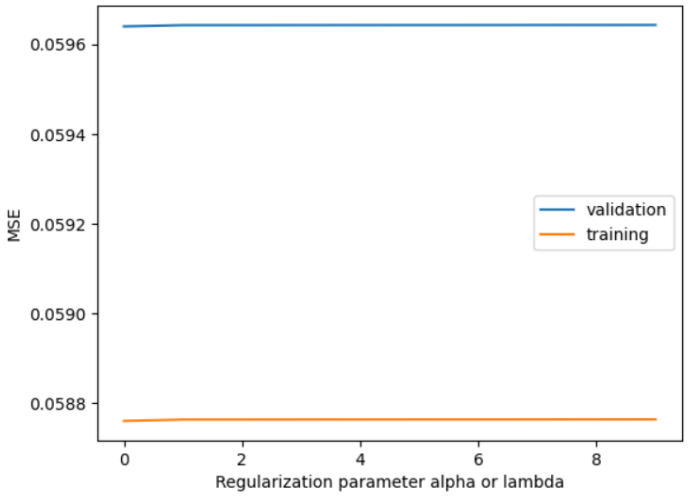

Fig. 45

In-sample MSE = 0.058763
Validation MSE = 0.05964
Out-sample MSE = 0.06604

Omega MSE:

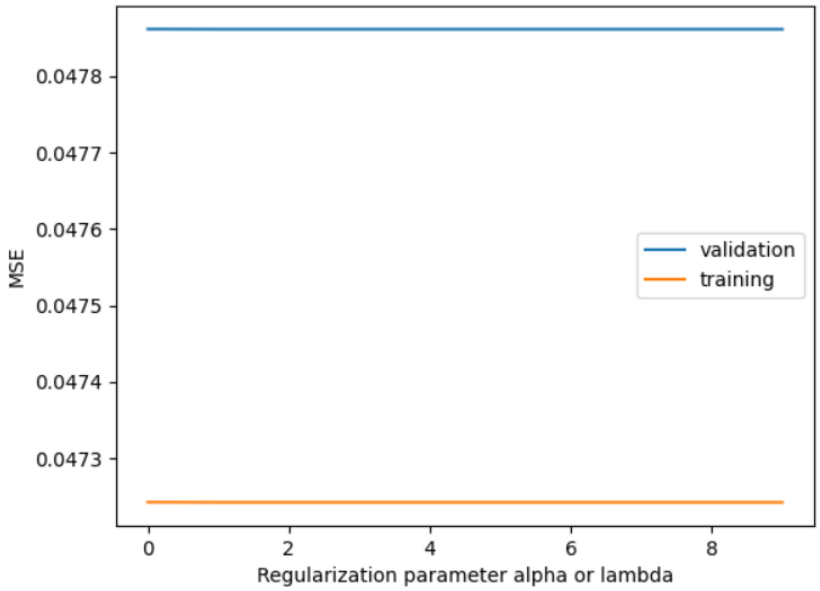

Fig. 46

In-sample MSE = 0.04724216267807056
Validation MSE = 0.05964313150902283
Out-sample MSE = 0.0564443296827880

Model 3: XG Boost

Velocity Prediction MSE:

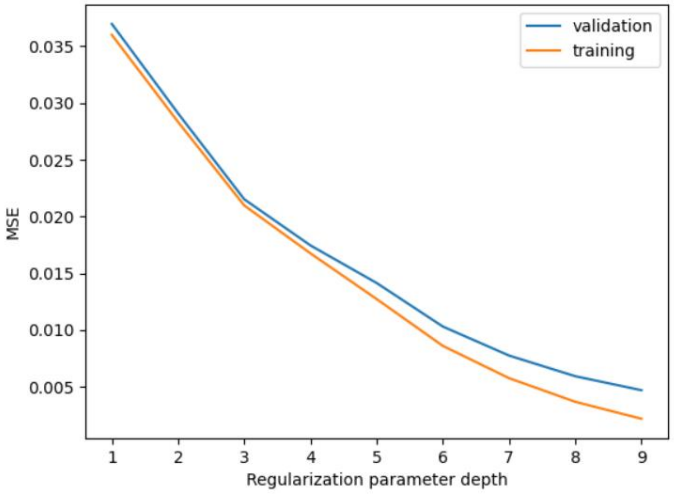

Fig. 47

In-sample MSE = 0. 00218
Validation MSE = 0.00319
Out-sample MSE = 0.0364

Omega:

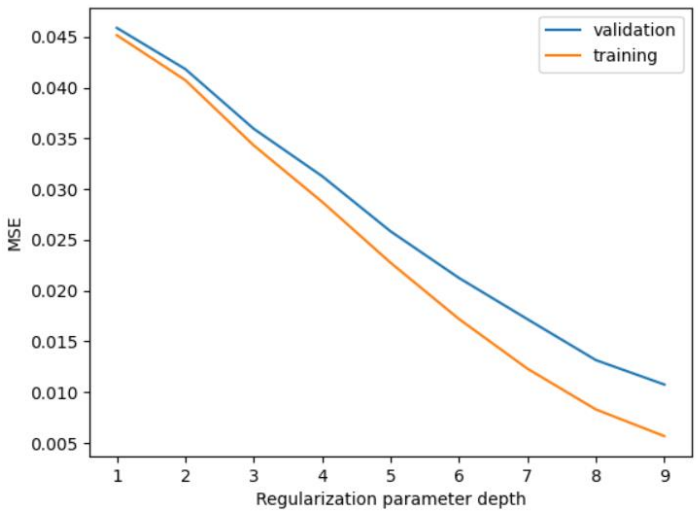

Fig. 48

In-sample MSE = 0. 0.005
Validation MSE = 0.01
Out-sample MSE = 0.035

Model 4: MLP

Velocity MSE:

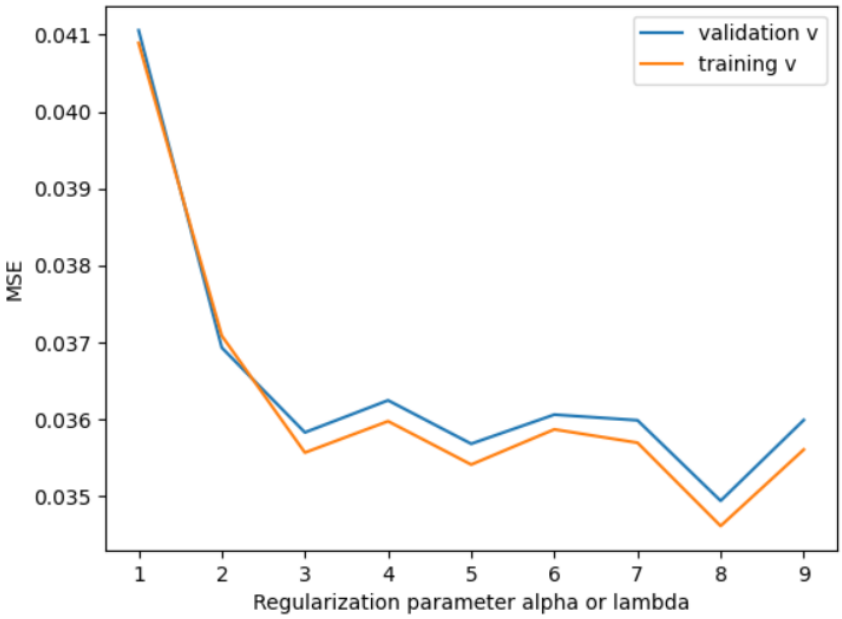

Fig. 49

In-sample MSE = 0.036
Validation MSE = 0.038
Out-sample MSE = 0.042

Omega:

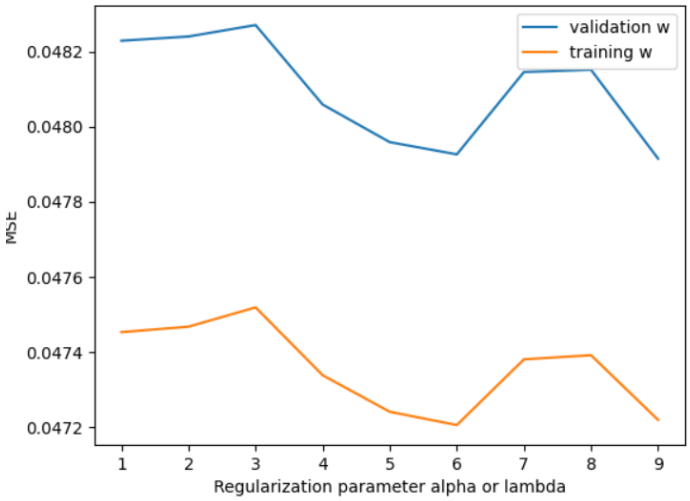

Fig. 50

UNIVERSITY OF MARYLAND

In-sample MSE = 0.0472
Validation MSE = 0.0480
Out-sample MSE = 0.0521

HENCE, THE PERFORMANCE IS BEST WITH XGBOOST!!