**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 1- CSV files that have more than enough data to be split in the training and validation set.

**Feature Selection:**

There are three main goals to feature selection. Improve the accuracy with which the model is able to predict for new data. Reduce computational cost. 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. The features defined are as follows:

* Distance from local goal position

IMAGES if possible

* Distance from final goal position
* Angular error in shortest trajectory to local goal and vehicle heading angle

A picture containing text, whiteboard

Description automatically generated

Fig. Angular error

* Angular error in shortest trajectory to final goal and vehicle heading angle
* Lidar output in LOS (Line of Sight) of the robot
* Lidar output towards the shortest view for the local goal
* Lidar output towards the shortest view for the final goal
* Maximum distance the robot can travel towards the shortest trajectory for the local goal

A whiteboard with writing on it

Description automatically generated with medium confidence

A whiteboard with writing on it

Description automatically generated with medium confidence

* Maximum distance the robot can travel towards the shortest trajectory for the final goal
* Note: The limitation in output velocity and omega will also be considered

There can be more such features

Model selection:

Linear Ridge Regression:

**Velocity Prediction:**

Parameter Tuning:

Graphical user interface

Description automatically generated with medium confidence

Learning Curve:

Chart

Description automatically generated

Scoring and error:

R2 scores:

|  |  |
| --- | --- |
| Validation Scores | |
| Regularization parameter alpha | Scores |
| 0 | 0.6125 |
| 0.2 | 0.6126 |
| 0.4 | 0.6126 |
| 0.6 | 0.6126 |
| 0.8 | 0.6126 |
| 0.9 | 0.6126 |

**Tuned parameter value = 0.9**

|  |  |
| --- | --- |
| Training Scores | |
| Regularization parameter alpha | Scores |
| 0 | 0.6272781927749702 |
| 0.2 | 0.6274001578826105 |
| 0.4 | 0.627400480084319 |
| 0.6 | 0.6274008021823565 |
| 0.8 | 0.6274011241552864 |
| 0.9 | 0.6274012851134265 |
|  |  |

**Out-sample/Testing Score:**

R2 score = 0.6074012851134265

**Omega** **Prediction**:

Learning Curve:

Chart

Description automatically generated

**SVM:**

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**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC) or [**SGDClassifier**](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGDClassifier) instead, possibly after a [**Nystroem**](https://scikit-learn.org/stable/modules/generated/sklearn.kernel_approximation.Nystroem.html#sklearn.kernel_approximation.Nystroem) transformer.

Hyperparameter tuning:

* RBF Kernel:

Regularization:

Plotted against 1/alpha

Chart, line chart

Description automatically generated

R2 Score for Velocity = 0.483131221616796

Learning Curve:

Chart

Description automatically generated

Omega:

Chart, line chart

Description automatically generated

Learning Curve for Omega:

Chart

Description automatically generated

Outsample score = 0.04

* **Linear Kernel**: was not able to solve
* **Poly Kernel**: degree = 3, iterated through regularization,

Regularization: Plotted against 1/alpha

Chart, line chart

Description automatically generated

Outsample R2 score = 0.10818291218618581

Learning Curve:

Chart

Description automatically generated

* Sigmoid Kernel:

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

**XGBOOST (Source: NVIDIA.com):**

XGBoost is an open-source software library that implements optimized distributed gradient boosting machine learning algorithms under the [**Gradient Boosting**](https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/) framework.

[XGBoost](https://xgboost.ai/), which stands for Extreme Gradient Boosting, is a scalable, distributed [**gradient-boosted**](https://en.wikipedia.org/wiki/Gradient_boosting)  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**](https://en.wikipedia.org/wiki/Ensemble_learning) similar to random forest, for classification and regression. Ensemble learning algorithms combine multiple machine learning algorithms to obtain a better model.

Diagram

Description automatically generated

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**](https://developer.nvidia.com/blog/gradient-boosting-decision-trees-xgboost-cuda/) 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.

**Learning Curve:**

**Velocity->NEEDS MORE DATA POINTS**

**Chart

Description automatically generated**

**Omega:**

**Chart, line chart

Description automatically generated**

**Hyperparamter Tuning:**

**Velocity->**

**Chart, line chart

Description automatically generated**

**In-sample R2 score = 0.9928270626691554**

**Validation R2 score = 0.9624411385324139**

**Out-sample R2 score = 0.8710081516705481**

**Omega:**

**Chart, line chart

Description automatically generated**

**In-sample R2 score = 0.9544640660728486**

**Validation R2 score = 0.7824411385324139**

**Out-sample R2 score = 0.12173614727621995**

**NEURAL NETWORK:**

**Multi-layer Perceptron (Source: scikit-learn.org)**

**Multi-layer Perceptron (MLP)** is a supervised learning algorithm that learns a function f(⋅):Rm → Ro by training on a dataset, where m is the number of dimensions for input and o is the number of dimensions for output. Given a set of features X=x1,x2,...,xm and a target y, it can learn a non-linear function approximator for either classification or regression. 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. Figure 1 shows a one hidden layer MLP with scalar output.