**ENPM808A FINAL PROJECT REPORT**

# **General Project Pipeline followed**

**Diagram

Description automatically generated**

**My codebase includes following classes implementation:**

1. **Data\_import.py –** Checks for csv in the input folder. Checks for any inconsisteCombines 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

**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.

**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

Diagram

Description automatically generated

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 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

Diagram

Description automatically generated

**Fig. 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

Diagram

Description automatically generated

**Fig. Obstacle detection cost**

Diagram

Description automatically generated

**Fig. 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

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.

**Metrics Considered**

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.

As such variance is dataset dependent, R2 may not be meaningfully comparable across different datasets. 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.

Note: when the prediction residuals have zero mean, the R2 score and the [Explained variance score](https://scikit-learn.org/stable/modules/model_evaluation.html#explained-variance-score) are identical.

If y^i is the predicted value of the i-th sample and yi is the corresponding true value for total n samples, the estimated R2 is defined as:

R2(y,y^)=1−∑i=1n(yi−y^i)2∑i=1n(yi−y¯)2

where y¯=1n∑i=1nyi and ∑i=1n(yi−y^i)2=∑i=1nϵi2.

SOURCE: <https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics>

**Model test with box data:**

**Linear Ridge Regression**

**Velocity Prediction:**

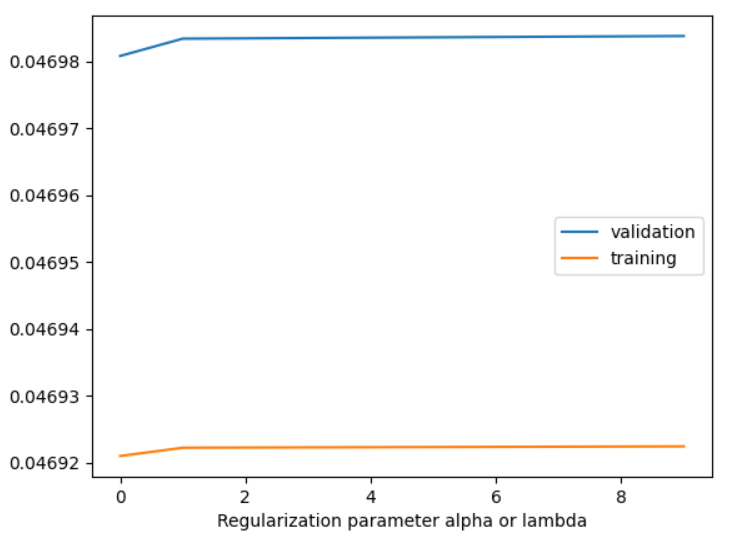
Parameter Tuning for R2 score and MSE:

* R2 Score

Graphical user interface

Description automatically generated with medium confidence

* MSE variation



**Large dataset**

**In-sample MSE = 0.046920975145010944**

**Validation MSE = 0.04698079021276492**

**Out-sample MSE = 0.051465936370142046**

Learning Curve:

Chart

Description automatically generated

Scoring and error:

**Omega** **Prediction**:

Learning Curve:

Chart

Description automatically generated

Hyperparamter tuning:

R2 score

Graphical user interface

Description automatically generated

But testing score is in negative

MSE:

Graphical user interface, application

Description automatically generated

**Large dataset omega**

**In-sample MSE = 0.06385259012498859**

**Validation MSE = 0.06472638682085763**

**Out-sample MSE = 0.06285082812775587**

With 150,000 data points:

Velocity learning curve:

A picture containing graphical user interface

Description automatically generated

Omega Learning Curve:

A picture containing graphical user interface

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 with poly degree 6**:

Regularization:

Plotted against 1/alpha

Chart, line chart

Description automatically generated

R2 Score for Velocity = 0.483131221616796

MSE:

Chart, line chart

Description automatically generated

**In-sample MSE = 0.044280547624567686**

**Validation MSE = 0.04472638682085763**

**Out-sample MSE = 0.04763863977038479**

Learning Curve:

Chart

Description automatically generated

Omega:

R2 Score

Chart, line chart

Description automatically generated

MSE:

Chart, line chart

Description automatically generated

**In-sample MSE = 0.05958532990772832**

**Validation MSE = 0.062082018320186366**

**Out-sample MSE = 0.06019050373651059**

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

Velocity:

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Learning Curve:

Chart

Description automatically generated

Outsample R2 score = 0.10818291218618581

Omega:

Chart, line chart

Description automatically generated

Learning curve:

A picture containing graphical user interface

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**

**Hyperparameter Tuning:**

**Velocity->**

**Chart, line chart

Description automatically generated**

**In-sample R2 score = 0.9928270626691554**

**Validation R2 score = 0.9624411385324139**

**Out-sample R2 score = 0.88950**

**Chart, line chart

Description automatically generated**

**Large Dataset**

**In-sample MSE = 0.001990886776502641**

**Validation MSE = 0.002237801974321346**

**Out-sample MSE = 0.013310271713501888**

**Omega:**

**R2 Score:**

**Chart, line chart

Description automatically generated**

**In-sample R2 score = 0.8807**

**Validation R2 score = 0.69446**

**Out-sample R2 score = 0.3016**

**MSE (Large dataset):**

**Chart, line chart

Description automatically generated**

**In-sample MSE = 0.007678344423281377**

**Validation MSE = 0.019899980335489773**

**Out-sample MSE = 0.04383341306052144**

**Velocity Learning curve**

**Chart, line chart

Description automatically generated**

**Chart, line chart

Description automatically generated**

**After increasing data points:**

**In-sample R2 score = 0.8807**

**Validation R2 score = 0.69446**

**Out-sample R2 score = 0.3016**

**reg:linear is now deprecated in favor of reg:squarederror.**

**[16:35:21] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-030221e36e1a46bfb-1/xgboost/xgboost-ci-windows/src/objective/regression\_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.**

**NEURAL NETWORKS**

**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.

**Diagram

Description automatically generated**

**Regularisation and hyperparameter tuning:**

**With the max\_iter parameter set to 500**

**Velocity:**

**Chart, line chart

Description automatically generated**

**In-sample R2 score = 0.6544640660728486**

**Validation R2 score = 0.5824411385324139**

**Out-sample R2 score = 0.5711991286745627**

**Omega:**

**Chart, line chart

Description automatically generated**

**In-sample R2 score = 0.0544**

**Validation R2 score = 0.0582**

**Out-sample R2 score = 0.052**

Learning Curve:

Velocity

Chart

Description automatically generated with low confidence

Omega:

A picture containing chart

Description automatically generated

After data scaling:

Velocity->

A picture containing chart

Description automatically generated

Chart, line chart

Description automatically generated

**In-sample R2 score = 0.830**

**Validation R2 score = 0.835**

**Out-sample R2 score = 0.821**

Omega->

Chart

Description automatically generated

Chart, line chart

Description automatically generated

**In-sample R2 score = 0.2230**

**Validation R2 score = 0.2035**

**Out-sample R2 score = 0.1931**

**After Large dataset**:

Velocity:

R2 score: Chart, line chart

Description automatically generated

**In-sample R2 score = 0.610**

**Validation R2 score = 0.615**

**Out-sample R2 score = 0.591**

**Chart, line chart

Description automatically generated**

**In-sample MSE = 0.04691647128188403**

**Validation R2 score = 0.04699860370372856**

**Out-sample R2 score = 0.019885021045387995**

Omega:

Chart, line chart

Description automatically generated

**In-sample R2 score = 0.038**

**Validation R2 score = 0.036**

**Out-sample R2 score = 0.03**

**Chart, line chart

Description automatically generated**

**In-sample MSE = 0.06203824884540909**

**Validation MSE = 0.06289030049585587**

**Out-sample MSE = 0.06952008743539813**

**Model test with corridor data:**

**Linear:**

**Velocity:**

**MSE:**

**Graphical user interface, application

Description automatically generated**

**In-sample MSE = 0.058763898617297965**

**Validation MSE = 0.05964313150902283**

**Out-sample MSE = 0.06604592908874049**

**Learning Curve:**

**A picture containing graphical user interface

Description automatically generated**

**Omega:**

**MSE**

**Graphical user interface, application

Description automatically generated**

**In-sample MSE = 0.04724216267807056**

**Validation MSE = 0.05964313150902283**

**Out-sample MSE = 0.0564443296827880**

**Learning Curve:**

**Chart

Description automatically generated**

**XG Boost**

**Velocity Prediction:**

**MSE:**

**Chart, line chart

Description automatically generated**

**In-sample MSE = 0. 002185511791884443**

**Validation MSE = 0.003195511791884443**

**Out-sample MSE = 0.03642986279028171**