ENPM 808A - Introduction to Machine Learning

Final Project – A car like Robot

Student Name:

Hritvik Choudhari (UID: 119208793)

Professor Name:

Dr. Nikhil Chopra

Table of Contents

I. Problem Statement

For the final project we have the recorded data from a non-holonomic car like robot, shown in the figure below.

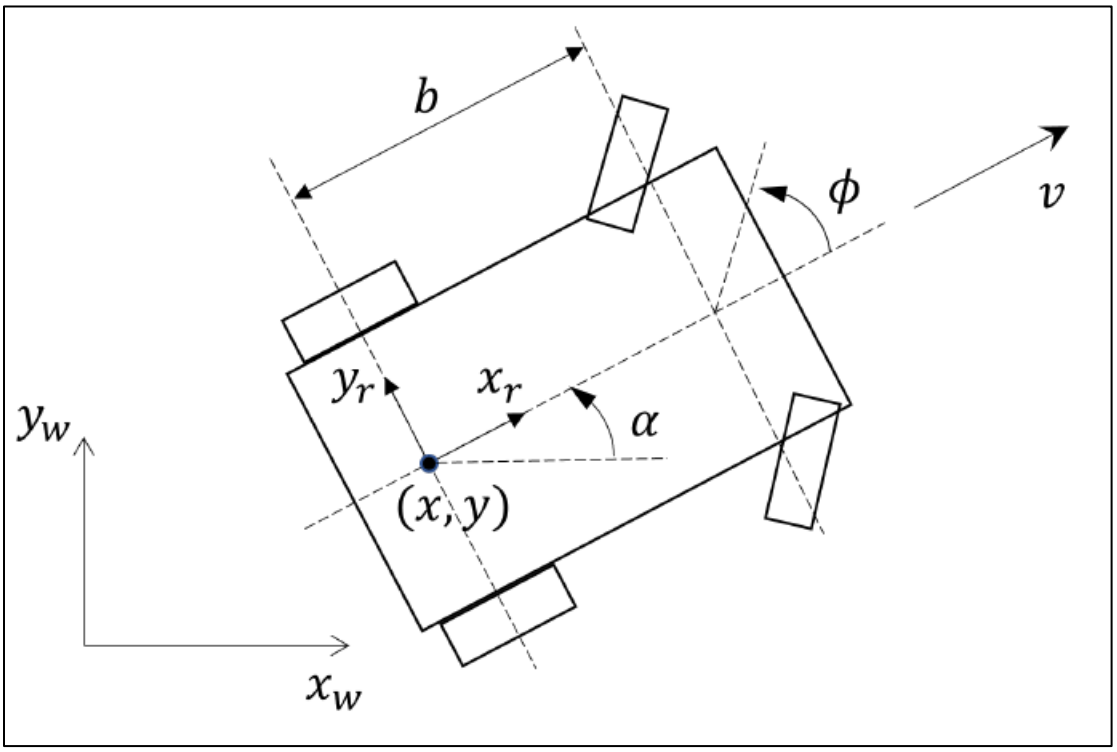


Figure 1: Car like Robot

$$\dot{x} = v\cos\alpha$$

$$\dot{y} = v\sin\alpha$$

$$\dot{\alpha} = \frac{v}{b}\tan\phi.$$

The car like robot senses its environment using a 1D LIDAR sensor. The data recorded is documented in the form of ".CSV" files. The LIDAR data is read and tabulated as columns of the csv file, total 1080 columns named from 'Laser 1' to 'Laser 1080' where each column is a 0.25-degree scan of the LIDAR. The final goal, local goal and robot's current position and pose information (which are spatially represented by 3D Cartesian coordinated and quaternion) is provided in subsequent columns. The last two columns consist of target variables linear velocities and the angular velocities of the robot. The data provided is captured from 3 different sets namely "Corridor", "Open_box" where the robot is tested under different conditions to make learning more general. The file "special" contains high instances of some special maneuvers such as backing up.

Data preprocessing for the input is done to reduce noise and computation time. Transforming or scaling the inputs and reducing the number of features will help us save a lot of computational time and the model will run more effectively giving us better results. In the preprocess step, the data from the above 3 sets were combined to form a complete training dataset. Similar process was done to obtain testing dataset. NaN values were also checked and if there were any, the entire sample was eliminated from the dataset using the dropna function.

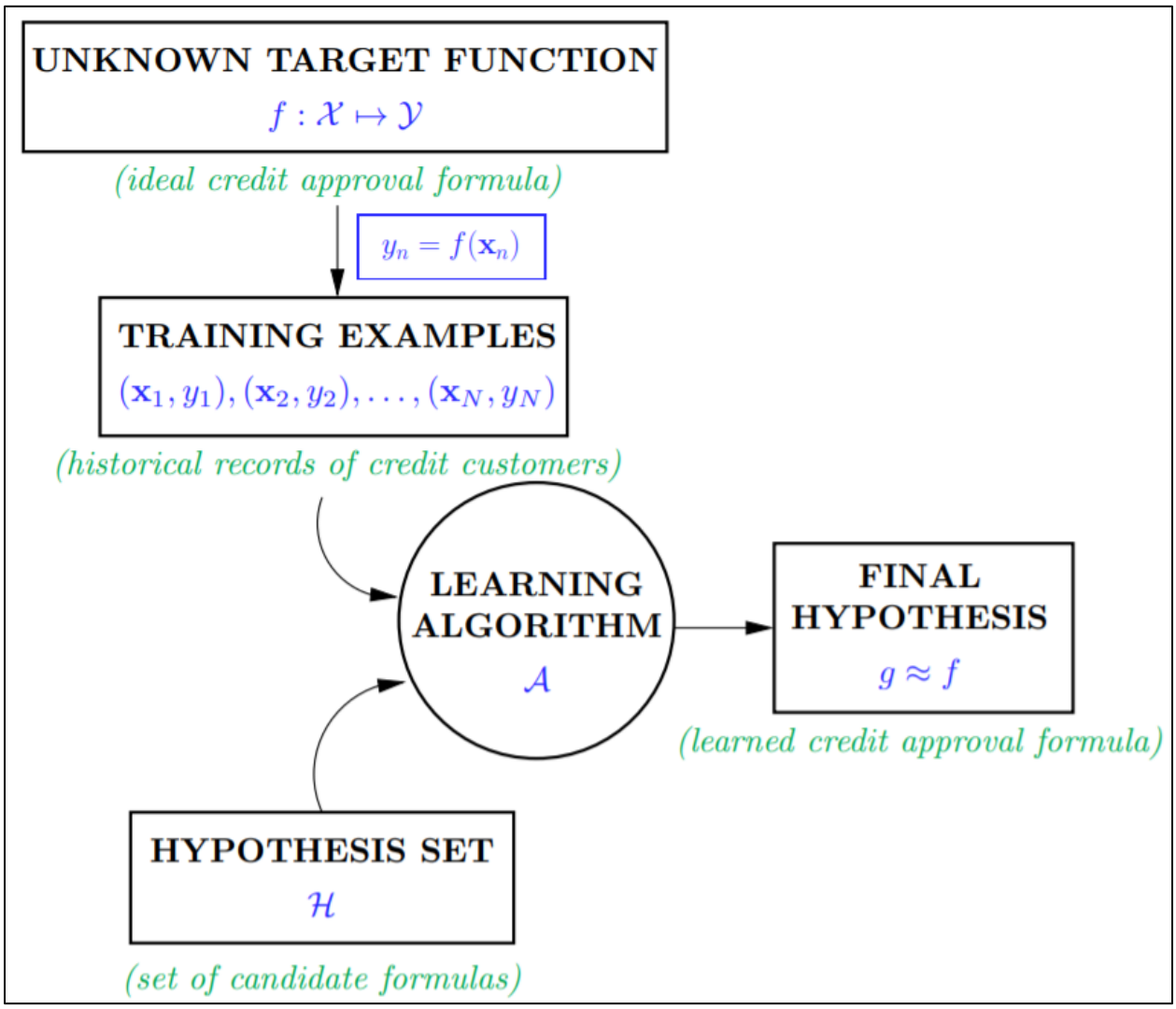
II. Feature Selection

The below code concatenates all the CSVs present in the selected folder into a single training dataset. Here we combine 20 columns of the LIDAR data into one by taking their mean which is equivalent to scanning at every 5 degrees. Since we know that quaternion q = qr + qi + qj + qk, we can add the qr and qk columns to a single column since qr and qi are equal to zero. This results in a CSV that has 60 features and last 2 columns as outputs or target features.

```
def angle_mean_data(df,one_scan_deg):
    new_df= pd.DataFrame()
    for i in range(len(df.iloc[:,:1081].columns)):
        if i%one_scan_deg != 0:
            pass
        else:
            add= df.iloc[:,i-one_scan_deg:i].mean(axis=1)
            new_df= pd.concat([new_df,add], axis=1, ignore_index= True)
    new_df.drop(0,axis=1, inplace= True)
    new_data= pd.concat([new_df,df.iloc[:,1080:]], axis=1)
    new_data.columns= col_names
    df.insert(loc=56, column='Final_goal_q', value= (new_data['Final_goal_qr'] + new_data['Final_goal_qk']))
    df.insert(loc=59, column='Local_goal_q', value= (new_data['Local_goal_qr'] + new_data['Local_goal_qk']))
    df.insert(loc=62, column='Robot_pos_q', value= (new_data['Robot_pos_qr'] + new_data['Robot_pos_qk']))
    new_data.drop(['Final_goal_qr','Final_goal_qk'], axis=1,inplace=True)
    new_data.drop(['Local_goal_qr','Local_goal_qk'], axis=1,inplace=True)
    new_data.drop(['Robot_pos_qr','Robot_pos_qk'], axis=1,inplace=True)
    return new_data
```

III. Model Training, Selection and Validation:

The below image shows the general machine learning pipeline that we follow to get the final hypothesis that is approximately close to the unknown target function.



This is a regression problem since our problem statement requires us to predict the linear and angular velocities. So, we went ahead to try 3 different models namely Linear Regression, Support vector machines and Neural networks.

A pipeline was made where the data was firstly standardized to improve the computation and accuracy of the results. As we have multiple outputs, MultiOutputRegressor function was used in case of Linear Regression and SVM models.

a) Validation and Hyper-parameter tuning for Linear Regression model.

In case of linear model, around 60000 datapoints(since the results were constant after certain data points and computing was high for more data) randomly from the training set was used for training and we selected the following hyper-parameters:

a) Penalty- [l1 , l2]
b) Alpha- [0.0001, 0.001, 0.01, 0.1] (Regularization term)
c) Learning rate- [constant, optimal, invscaling, adaptive]

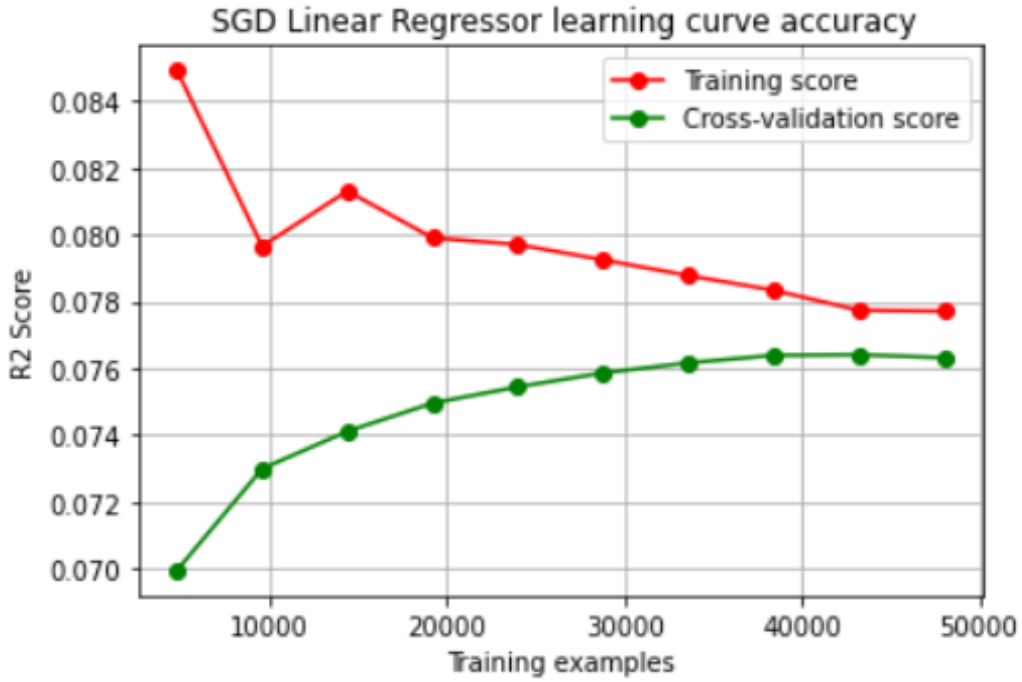
Hyper-parameter tuning was performed where we selected among random 20 combinations of the above parameters using RandomizedSearchCV function. From this the best estimator and parameter was obtained which was. The optimal hyper parameter obtained were:

- Best penalty- l2
- Best alpha- 0.01
- Best learning rate- adaptive

Later the learning curve was obtained with scoring parameter of 'R2' along with the validation curves for each hyper-parameter. Below are the code for this model along with the results:

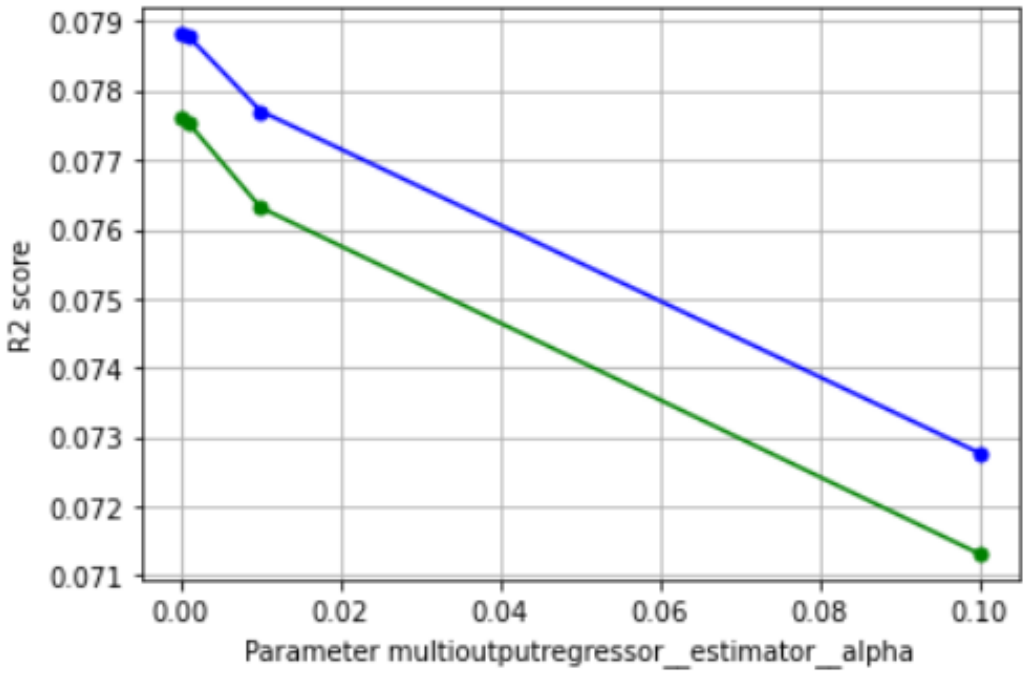
```
def best_LR(X_train, Y_train):
    model = dict(estimator= make_pipeline(StandardScaler(),MultiOutputRegressor(estimator= SGDRegressor(max_iter= 10000))),
    parameter={
    #loss = ['squared_error', 'epsilon_insensitive','squared_epsilon_insensitive'],
    "multioutputregressor__estimator__penalty" : ['l1', 'l2'],
    "multioutputregressor__estimator__alpha" : [0.0001, 0.001, 0.01, 0.1],
    "multioutputregressor__estimator__learning_rate" : ['constant', 'optimal', 'invscaling', 'adaptive'],
    #eta0 = [1, 10, 100]
    })
    [estimator, params] = hyper_parameter_tuning(model, X_train, Y_train)
    plot_curve_learning(estimator, X_train, Y_train, " SGD Linear Regressor learning curve accuracy")
    plot_curve_validation_lr(model,estimator,params, X_train, Y_train, " SGD Linear Regressor validation curve accuracy")
    cv_results = cross_val_score(estimator, X_train, Y_train, cv=2, scoring='r2')
    print(f'\nLR: \n\tOptimal Hyper-parameters: {params} \n\tscores: {cv_results} \n\tmean score: {cv_results.mean()}')
    return [estimator, cv_results.mean()]
best_LR(X_train,y_train)
```

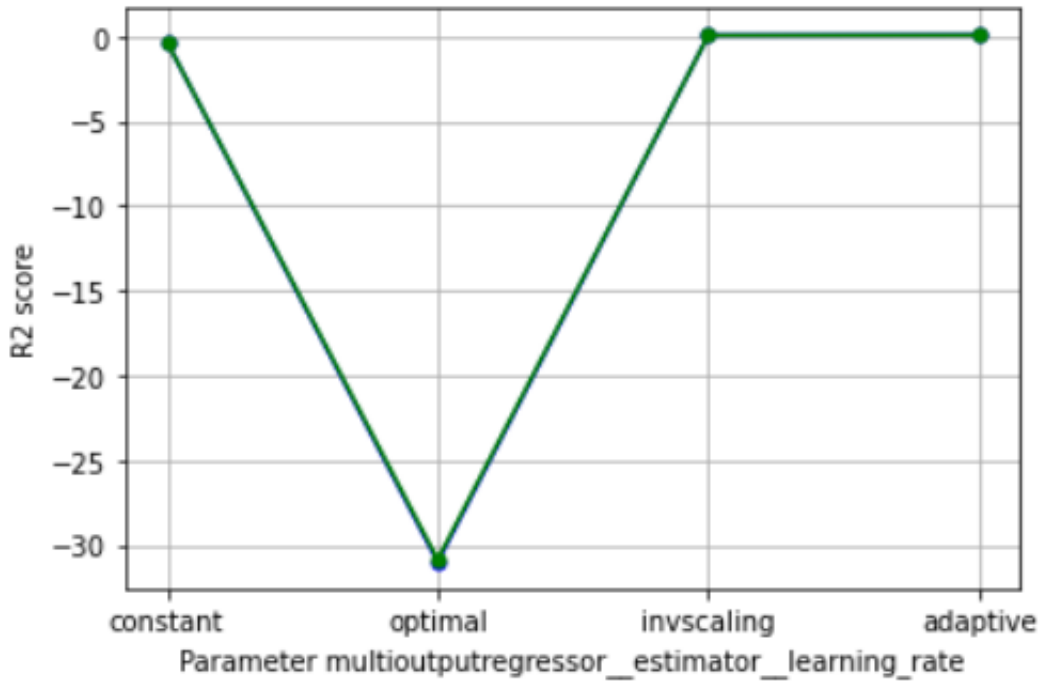
```
Fitting 2 folds for each of 20 candidates, totalling 40 fits
```

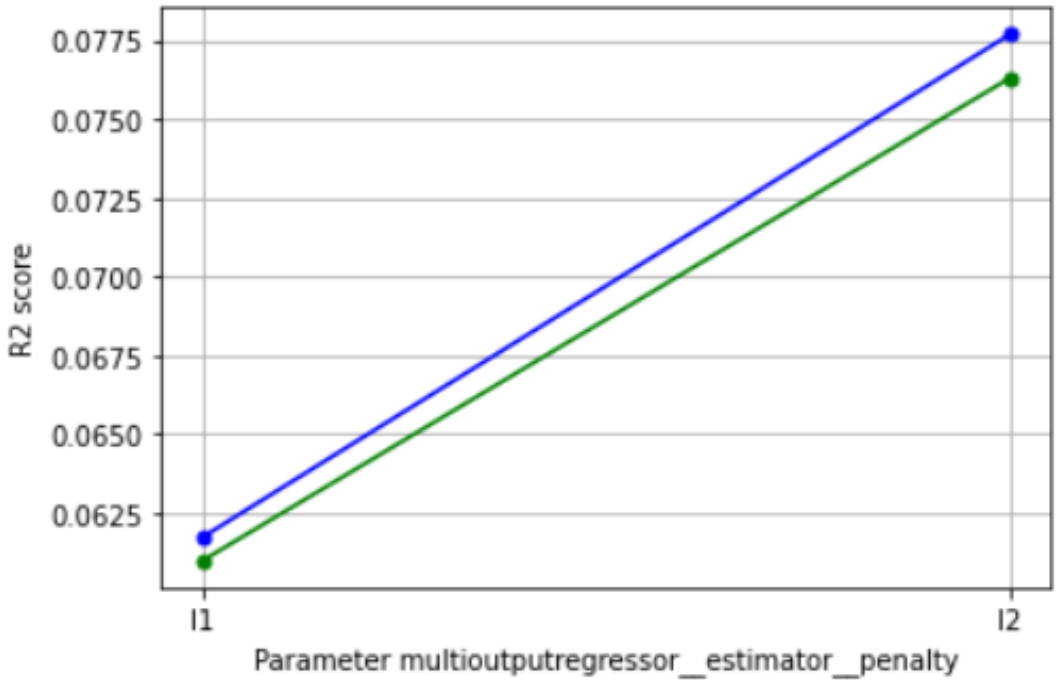



```
LR:
       Optimal Hyper-parameters: {'multioutputregressor__estimator__penalty': 'l2', 'multioutputregressor__estimator__learning
_rate': 'adaptive', 'multioutputregressor__estimator__alpha': 0.01}
       scores: [0.06114418 0.06312714]
       mean score: 0.06213565860002873

[Pipeline(steps=[('standardscaler', StandardScaler()),
                 ('multioutputregressor',
                  MultiOutputRegressor(estimator=SGDRegressor(alpha=0.01,
                                                              learning_rate='adaptive',
                                                              max_iter=10000)))]),
 0.06213565860002873]
```

Validation Curves:







b) Validation and Hyper-parameter tuning for SVM model.

For the case of SVM model, around 60000 datapoints(since the results were constant after certain data points and computing was high for more data) randomly from the training set was used for training and we selected the following hyper-parameters:

a) C- [1, 10, 100] (Regularization parameter)
b) Gamma- [0.0001, 0.01, 1]
c) Kernel - [rbf, poly]
d) Degree- [1, 2]

Hyper-parameter tuning was performed where we selected among random 20 combinations of the above parameters using RandomizedSearchCV function. From this the best estimator and parameter was obtained which was. The optimal hyper parameter obtained were:

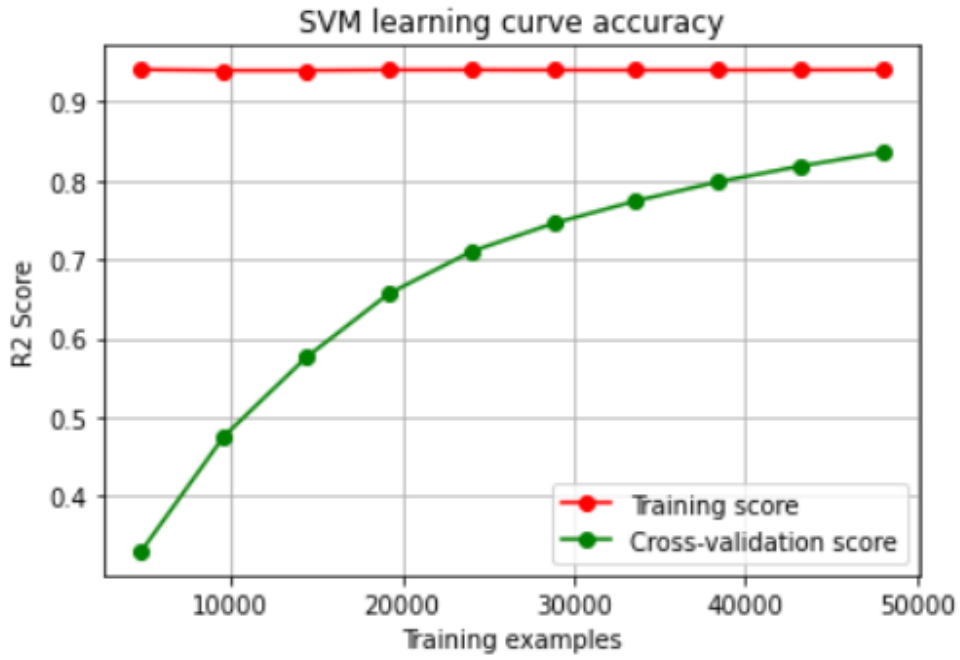
- Best C- 100
- Best gamma - 1
- Best kernel- rbf

Later the learning curve was obtained with scoring parameter of 'R2' along with the validation curves for each hyper-parameter. Below are the code for this model along with the results:

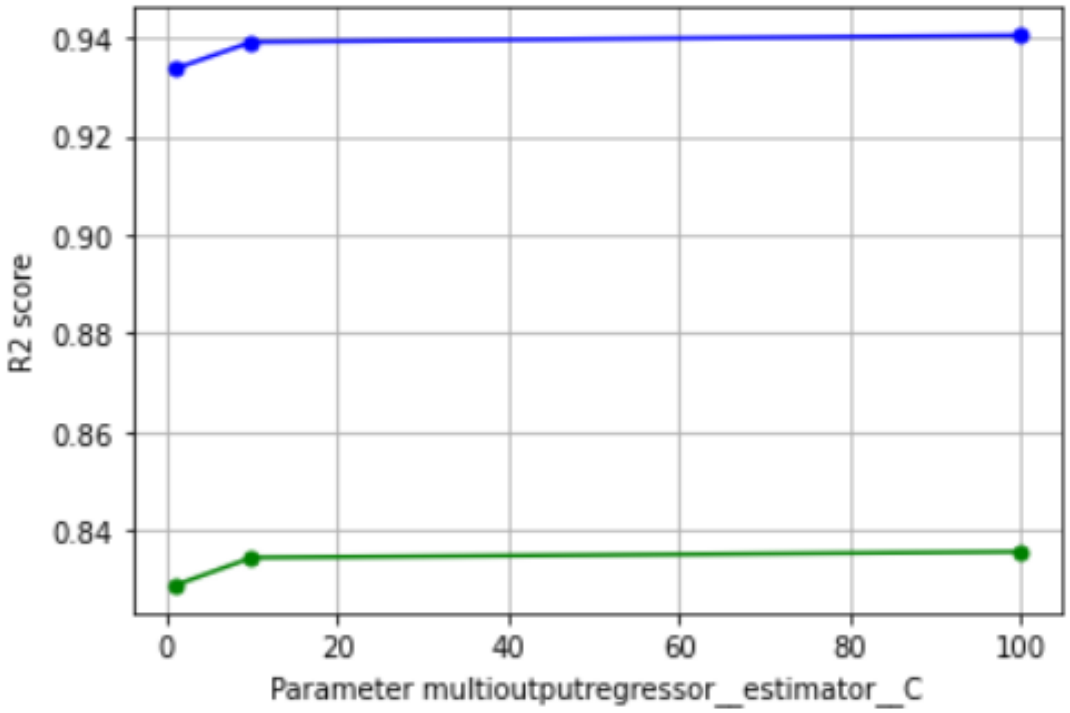
```
def best_SVM(X_train, Y_train):
    model = dict(estimator= make_pipeline(StandardScaler(),MultiOutputRegressor(estimator=SVR(cache_size=1000))),
    parameter=[{
    'multioutputregressor__estimator__C': [1, 10, 100],
    'multioutputregressor__estimator__gamma': [0.0001, 0.01, 1],
    'multioutputregressor__estimator__kernel': ['rbf']
    },
    {
    'multioutputregressor__estimator__C': [1, 10, 100],
    'multioutputregressor__estimator__degree': [1, 2],
    'multioutputregressor__estimator__kernel': ['poly']
    }])
    [estimator, params] = hyper_parameter_tuning(model, X_train, Y_train)
    plot_curve_learning(estimator, X_train, Y_train, "SVM learning curve accuracy")
    plot_curve_validation_svm(model,estimator,params, X_train, Y_train, " SGD Linear Regressor validation curve accuracy")
    cv_results = cross_val_score(estimator, X_train, Y_train, cv=2, scoring='r2')
    print(f'\nSVM: \n\tOptimal Hyper-parameters: {params} \n\tscores: {cv_results} \n\tmean score: {cv_results.mean()}')
    return [estimator, cv_results.mean()]

best_SVM(X_train,y_train)
```

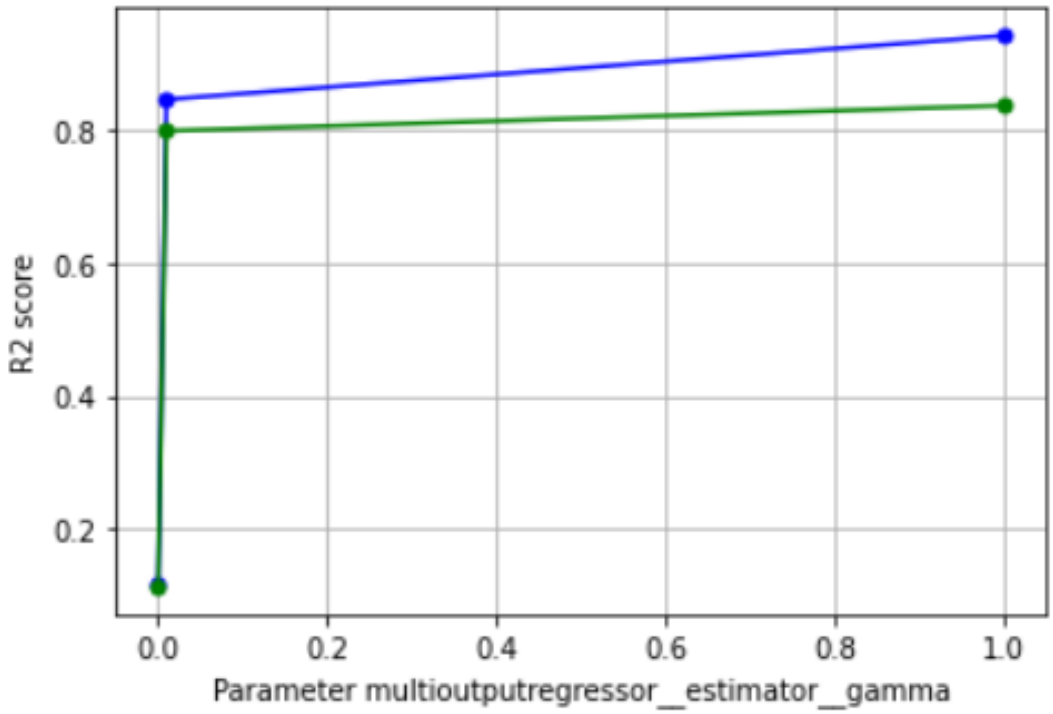
```
SVM:
        Optimal Hyper-parameters: {'multioutputregressor__estimator__kernel': 'rbf', 'multioutputregressor__estimator__gamma':
1, 'multioutputregressor__estimator__C': 100}
        scores: [0.73599831 0.70817686]
        mean score: 0.7220875816047061

[Pipeline(steps=[('standardscaler', StandardScaler()),
                 ('multioutputregressor',
                  MultiOutputRegressor(estimator=SVR(C=100, cache_size=1000,
                                                     gamma=1)))]),
 0.7220875816047061]
```

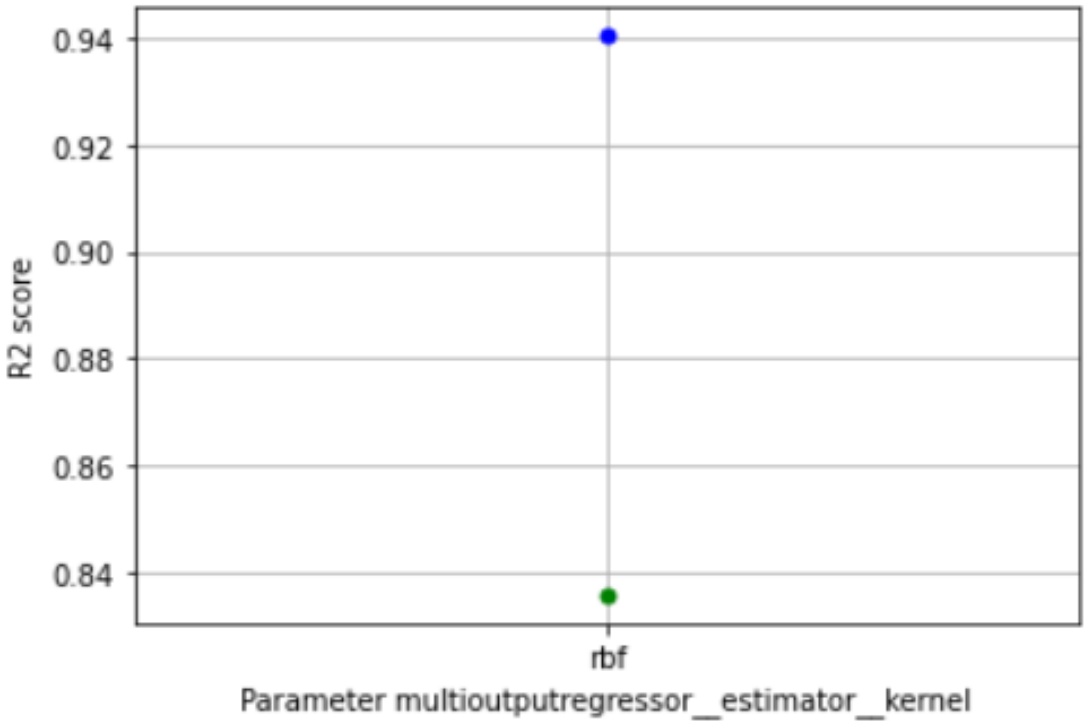
```
Fitting 2 folds for each of 15 candidates, totalling 30 fits
```



Validation Curves:




R2 score
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0
Parameter multioutputregressor__estimator__gamma


R2 score
0.94
0.92
0.90
0.88
0.86
0.84
rbf
Parameter multioutputregressor__estimator__kernel

c) Validation and Hyper-parameter tuning for Neural network model.

For the case of NN model, around 600000 datapoints randomly from the training set was used for training and we selected the following hyper-parameters:

a) solver- [adam, sgd] (Strength of the L2 regularization term)
b) activation- [relu, tanh]
c) alpha - [[0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009]]
d) hidden layers sizes- range(5,100)

Hyper-parameter tuning was performed where we selected among random 20 combinations of the above parameters using RandomizedSearchCV function. From this the best estimator and parameter was obtained which was. The optimal hyper parameter obtained were:

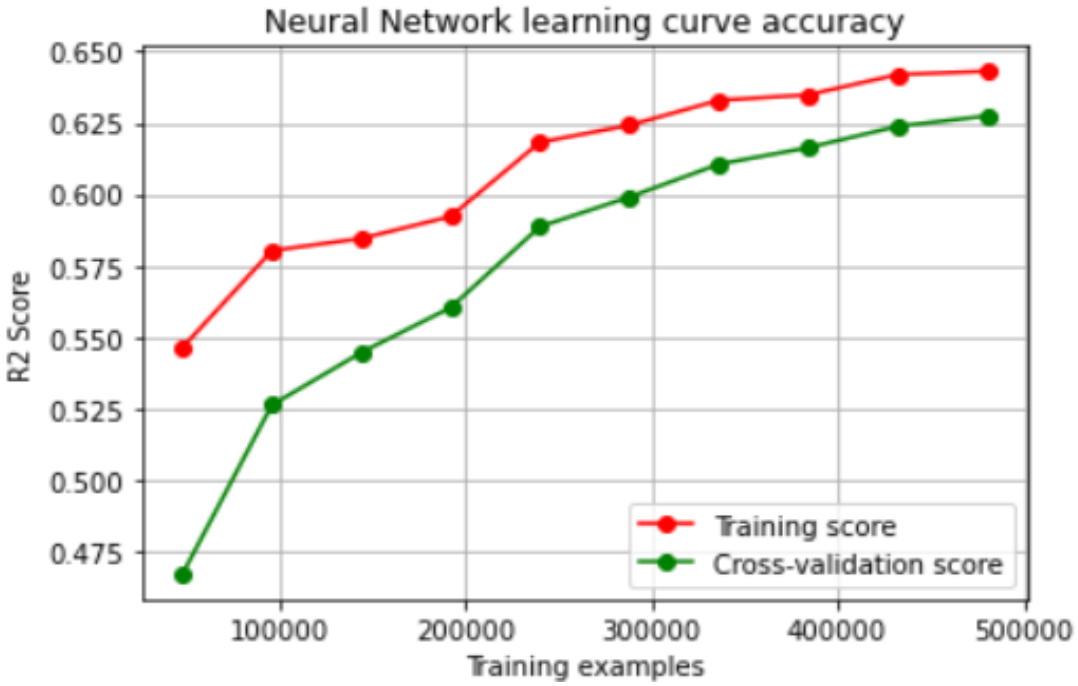
- Best solver - sgd
- Best activation - relu
- Best alpha- 0.002
- Best hidden layer size- (77,77,77)

Later the learning curve was obtained with scoring parameter of 'R2' along with the validation curves for each hyper-parameter. Below are the code for this model along with the results:

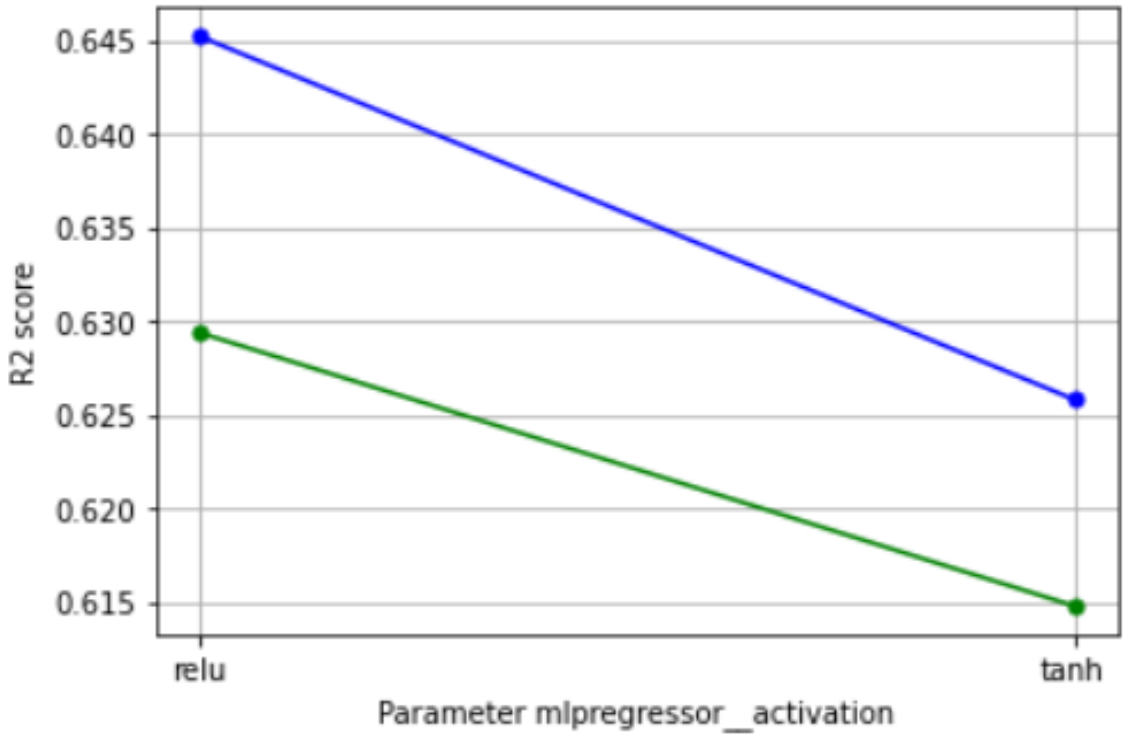
```python
def best_NN(X_train, Y_train, kfold):
    model = dict(estimator= make_pipeline(StandardScaler(),MLPRegressor(learning_rate_init= 0.01)),
    parameter={
    'mlpregressor__solver': ['adam','sgd'],
    'mlpregressor__activation': ['relu','tanh'],
    'mlpregressor__alpha': np.concatenate(
            (
            np.arange(0.001,0.01,0.001),
            np.arange(0.01,0.01,0.01),
            np.arange(0.1,0.1,0.1)
            )
        ),
        'mlpregressor__hidden_layer_sizes':
            [(i,) for i in np.arange(5,100,1)]+
            [(i,i,) for i in np.arange(5,100,1)]+
            [(i,i,i) for i in np.arange(5,100,1)],
    })
    [estimator, params] = hyper_parameter_tuning(model, X_train, Y_train)
    plot_curve_learning(estimator, X_train, Y_train, "Neural Network learning curve accuracy")
    plot_curve_validation_nn(model,estimator,params, X_train, Y_train, " Neural Network validation curve accuracy")
    cv_results = cross_val_score(estimator, X_train, Y_train, cv=kfold, scoring='r2')
    print(f'\nNN: \n\tOptimal Hyper-parameters: {params} \n\tscores: {cv_results} \n\tmean score: {cv_results.mean()}')
    return [estimator, cv_results.mean()]

best_NN(X_train_nn, y_train_nn, 2)
```

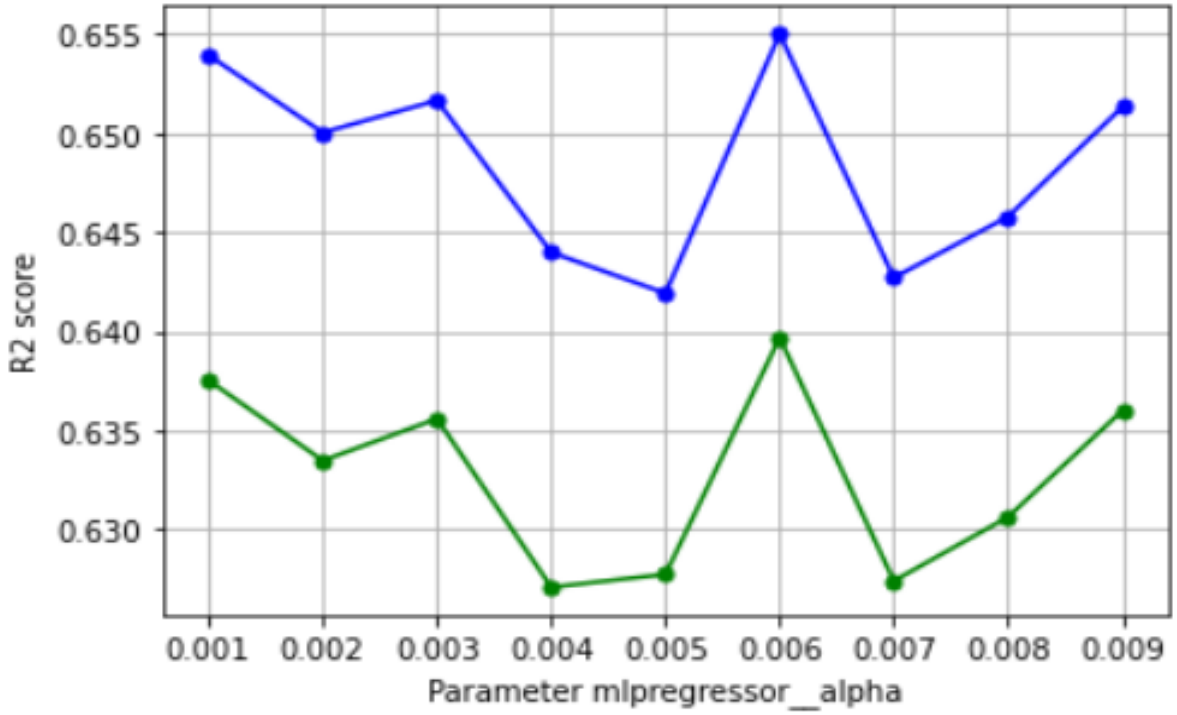
```
NN:
       Optimal Hyper-parameters: {'mlpregressor__solver': 'sgd', 'mlpregressor__hidden_layer_sizes': (77, 77, 77), 'mlpregress
or__alpha': 0.002, 'mlpregressor__activation': 'relu'}
       scores: [0.51787369 0.52232602]
       mean score: 0.5200998560333945

[Pipeline(steps=[('standardscaler', StandardScaler()),
                 ('mlpregressor',
                  MLPRegressor(alpha=0.002, hidden_layer_sizes=(77, 77, 77),
                               learning_rate_init=0.01, solver='sgd'))]),
 0.5200998560333945]
```

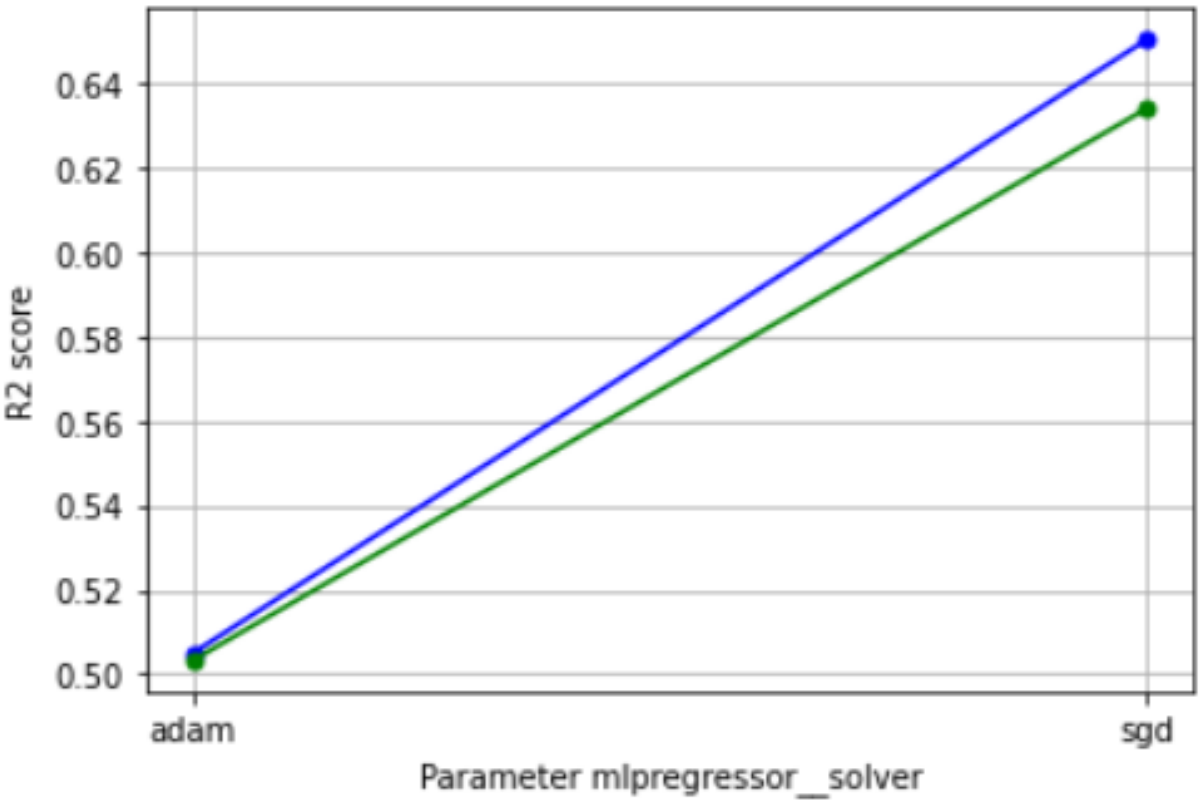
```
Fitting 2 folds for each of 20 candidates, totalling 40 fits
```



Validation Curves:




0.655
0.650
0.645
0.640
0.635
0.630
R2 score
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
0.009
Parameter mlpregressor__alpha


0.64
0.62
0.60
0.58
0.56
0.54
0.52
0.50
R2 score
adam
sgd
Parameter mlpregressor__solver

IV. Testing the selected model

From the cross-validation score results obtained we have selected SVM as our optimum model with optimal hyper-parameters obtained during training, as it provided the best cross validation score (0.72) among the three. Now we deploy it for testing over the test dataset where similar procedure for data preprocessing is done and then fed to the model. We randomly shuffled the training data and took 800000 data points for the model testing. The scoring parameter used to find E_test is "r_2":

```
# Model testing
test_model= make_pipeline(StandardScaler(),MultiOutputRegressor(estimator=SVR(kernel= 'rbf', gamma=1, C=100, cache_size=1000)))
test_model.fit(X_test, y_test)
y_pred = test_model.predict(X_test)
score= test_model.score(X_test,y_test)
print('E_test : ',1- score)
```

E_test: 0.073

V. Out-Sample error

We know the out of sample error is given by the Hoeffding's inequality (M=1):

$$E_{out} \leq E_{test} + \sqrt{\left(\frac{1}{2N} * log\left(\frac{(2M)}{\delta}\right)\right)}$$

Using this equation to find the Eout for the test data the code will be:

```
def E_out(score,N,confidence):
    a= 1-score
    b= np.sqrt((1/(2*N)* np.log(2/confidence)))
    return a+b
E_out(score,len(X_test),0.1)
```

```
0.08006603645800807
```

E_out: 0.08

Therefore, the E_out found is approximately equal to the E_test found earlier. This means that we have learnt something outside outside the dataset we have used.

VI. Conclusion

Hence, we successfully compared 3 different models, decided the better model, and trained a SVM model for predicting the linear and angular velocities of the test data with minimal E_in. This model can be used for predicting future values of linear velocity and angular velocity albeit with lower E_out.