Regression - TSK Models

Problem Description

In this project, we are tasked with implementing TSK (Takagi-Sugeno-Kang) fuzzy systems, solving regression problems.

This project is two fold. First we are given a simple dataset called '*Airfoil Self-Noise dataset*'. We then will be training 4 specific models with different configurations and evaluating their performance.

	Number of Membership Functions	Output Format
TSK_model_1	2	Singleton
TSK_model_2	3	Singleton
TSK_model_3	2	Polynomial
TSK_model_4	3	Polynomial

Figure 1: Classification of models to be trained.

Fig. 1 Models to be trained

The second part, includes a more complex, high-dimensional dataset '*Superconductivity dataset*'. Given its nature, before implementing any model, we are required to reduce its dimensions and fuzzy set of rules. We will then train the model, and assess its performance.

For both implementations, we will split the datasets in a 60-20-20 ratio for training-validating-checking. Each model's performace, learning curve and prediction errors will be visualised and analyzed.

TSK Application on Simple Dataset

Airfoil Self-Noise Dataset Overview

In this section, we will concentrate on the '*Airfoil Self-Noise dataset*', which comprises of 1503 data points, with 5 features and 1 output each.

Specifically the data points have the following features:

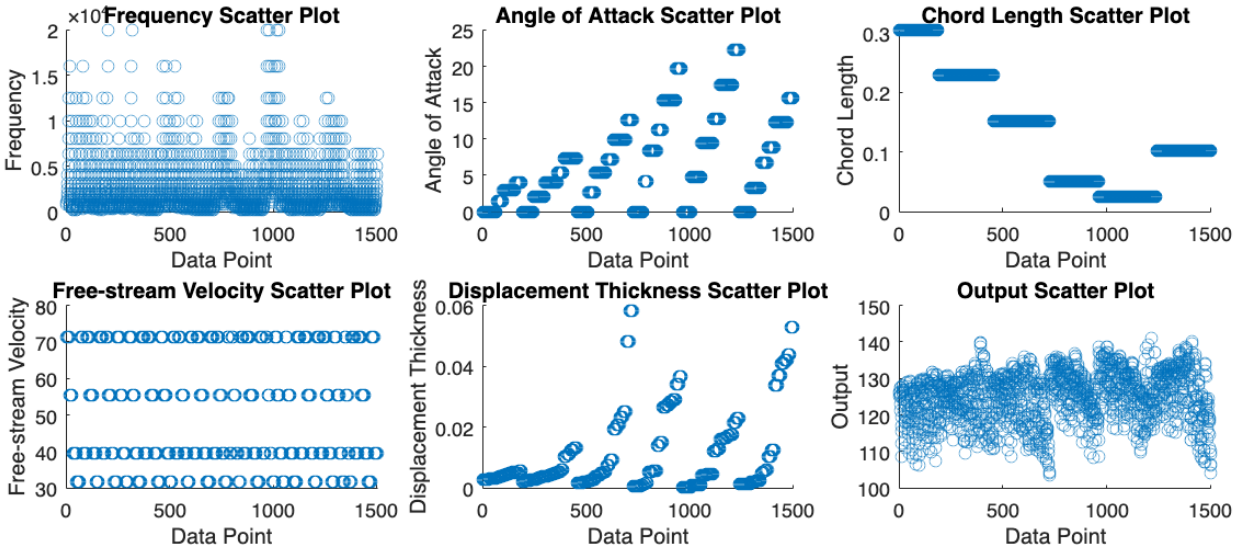
Input:

- f: Frequency $[\mathrm{Hz}]$
- alpha: Angle of attack $[^\circ]$
- c: Chord length $[m]$
- U_infinity: Free-stream velocity $\left[\frac{m}{s}\right]$
- delta: Suction side displacement thickness $[m]$

Output:

- SSPL: Scaled sound pressure level $[\mathrm{dB}]$

```
clc; clearvars; close all;
warning('off','all')
addpath("lib")
addpath("lib/datasets")
data_1 = table2array(readtable("datasets/airfoil_self_noise.dat"));

% Visualize Data
feature_names_1 = {'Frequency', 'Angle of Attack', 'Chord Length', 'Free-
stream Velocity', 'Displacement Thickness'};
visualize_data(data_1, feature_names_1, 1:5);
```



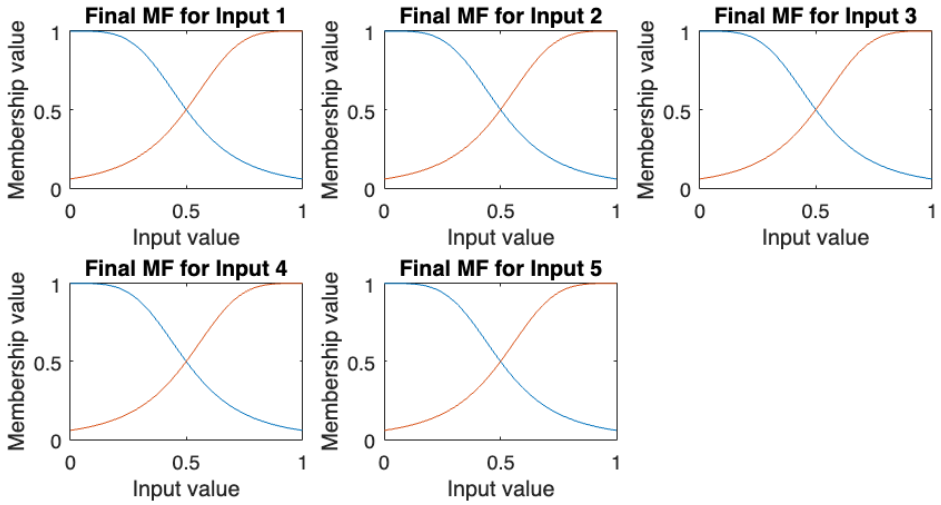

```
[Dtrn_1, Dval_1, Dchk_1] = split_data(data_1);
```

```
Dimensions of Training Data:
   902     6

Dimensions of Validation Data:
   300     6

Dimensions of Checking Data:
   301     6
```

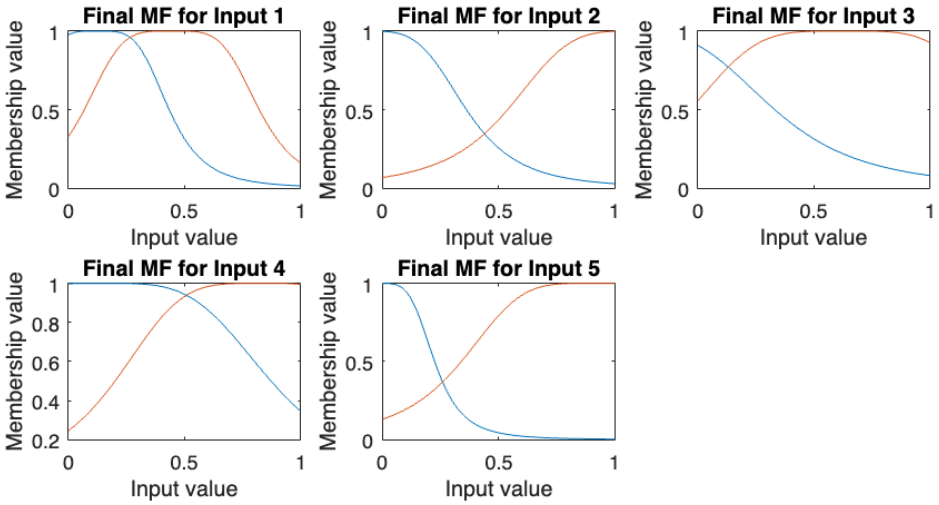
TSK Model 1

We will begin with implementing the first model, according the the requirements.

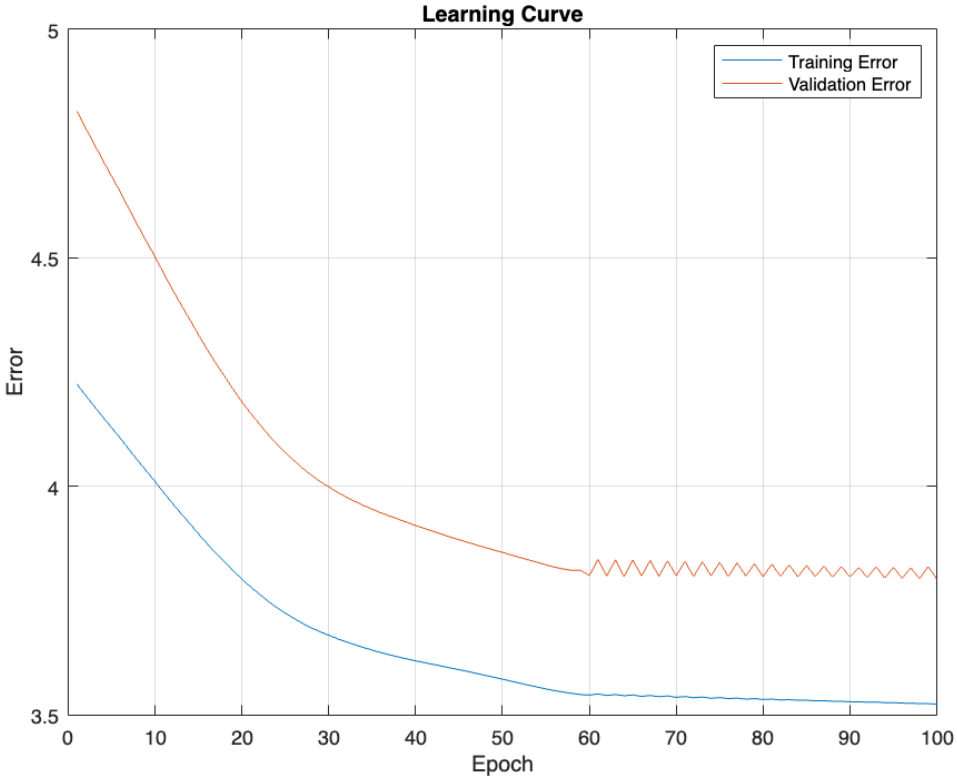
```
% Define configuration for Model 1
model_1 = struct('Part', '1','NumMFs', 2, 'InputMFT', 'gbellmf',
'OutputMFT', 'constant');

% Initialize Model 1
initial_fis_1 = initialize_tsk_model(Dtrn_1, model_1);
plot_mf(initial_fis_1, 5);
```

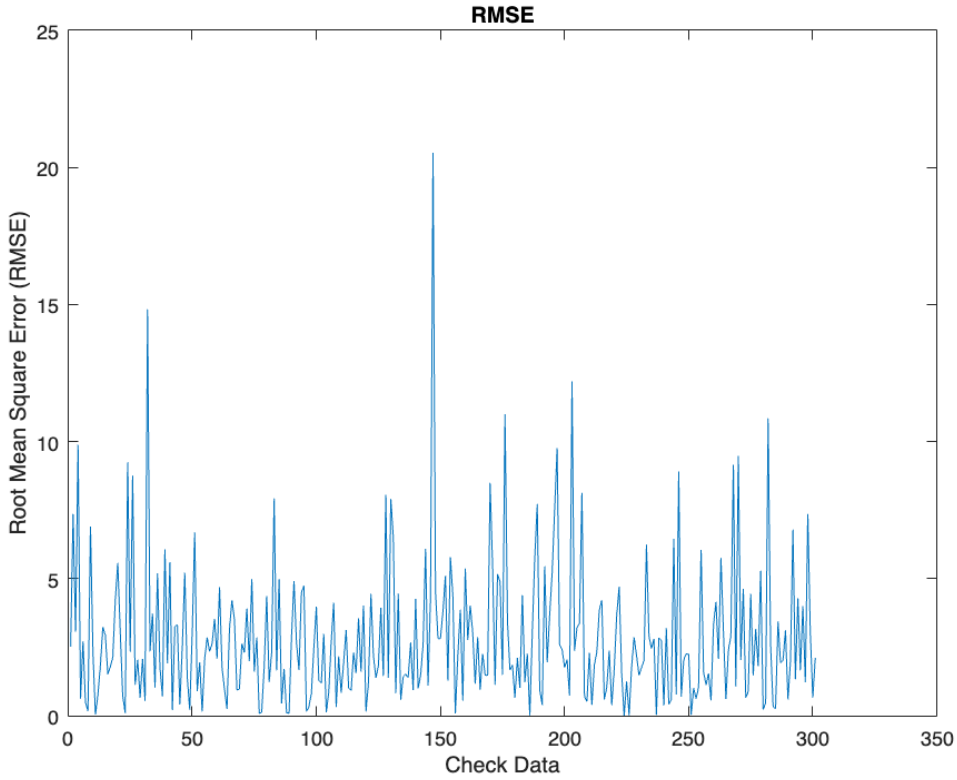


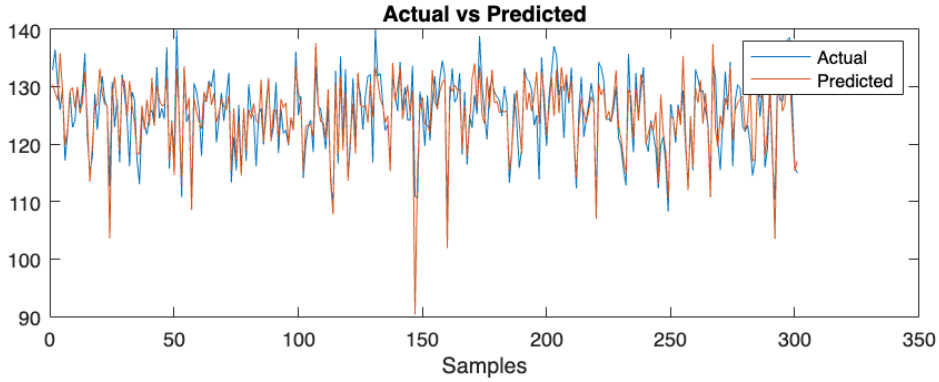

```
% Train Model 1
tic;
[trained_fis_1, training_info_1] = train_tsk_model(initial_fis_1, Dtrn_1,
Dval_1, 100);
time_1 = toc;
plot_mf(trained_fis_1, 5);
```

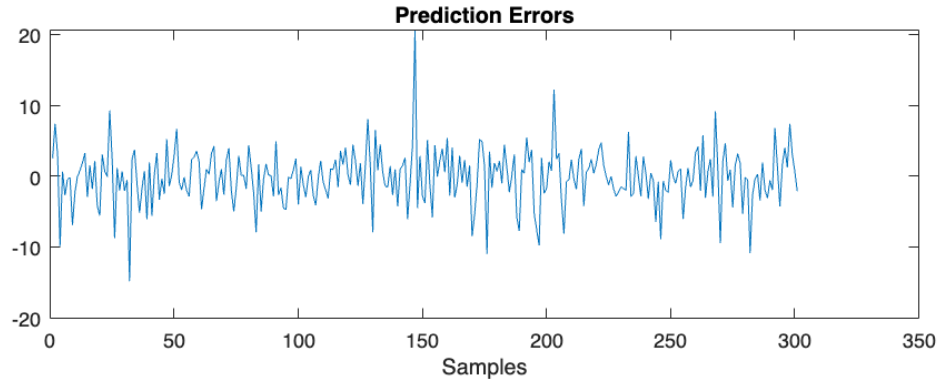



```
% Evaluate Model 1
[Rmse1, Nmse1, Ndei1, R21] = metrics_evaluation(trained_fis_1, Dchk_1);
plot_metrics(Dchk_1(:, end), evalfis(trained_fis_1, Dchk_1(:, 1:end-1)),
training_info_1.trainError, training_info_1.chkError);
```


Learning Curve
Training Error
Validation Error
Error
Epoch
5
4.5
4
3.5
0
10
20
30
40
50
60
70
80
90
100


RMSE
Root Mean Square Error (RMSE)
Check Data
25
20
15
10
5
0
0
50
100
150
200
250
300
350





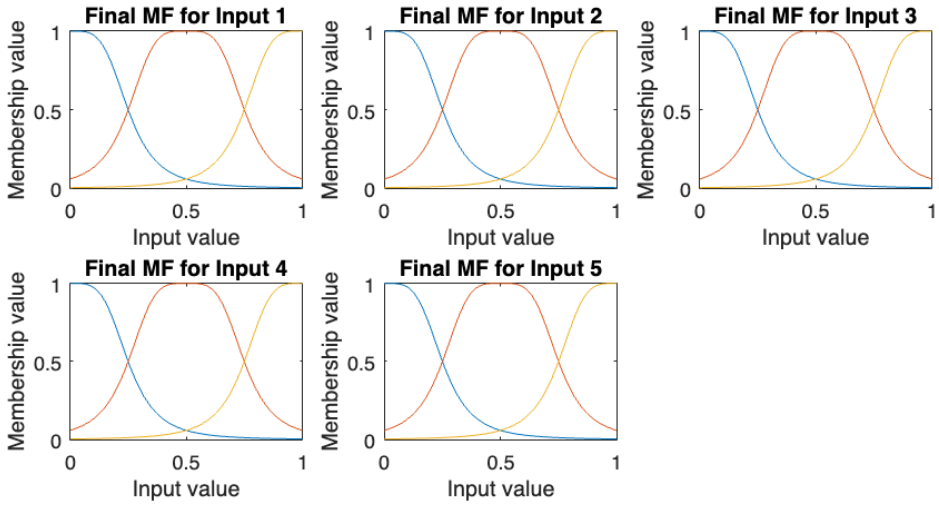

```
model_1_metrics = [Rmse1; Nmse1; Ndei1; R21];
fprintf('Error metrics for TSK model:\n RMSE: %.4f, NMSE: %.4f, NDEI: %.4f,
R2: %.4f\n', Rmse1, Nmse1, Ndei1, R21);
```

```
Error metrics for TSK model:
 RMSE: 3.8397, NMSE: 0.3379, NDEI: 0.5813, R2: 0.6621
```

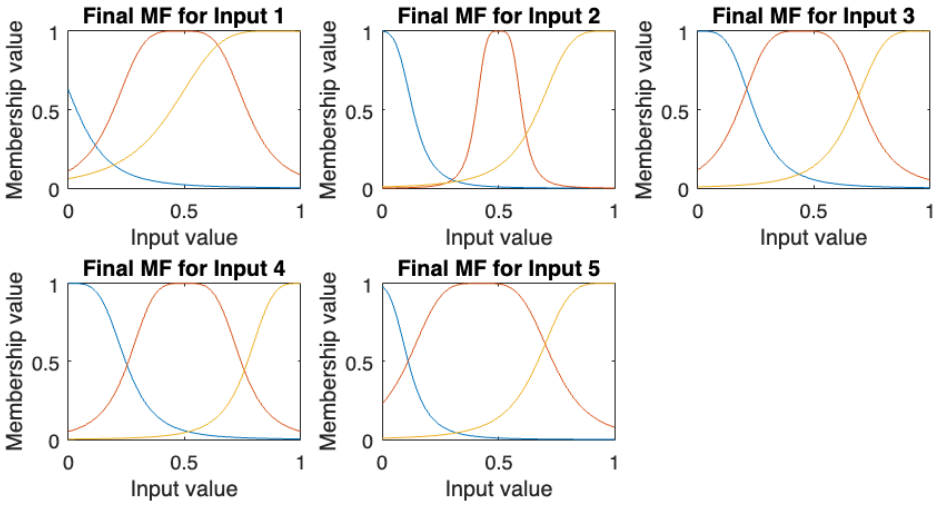
TSK Model 2

Based on the implementation of the first, we will continue with the second model.

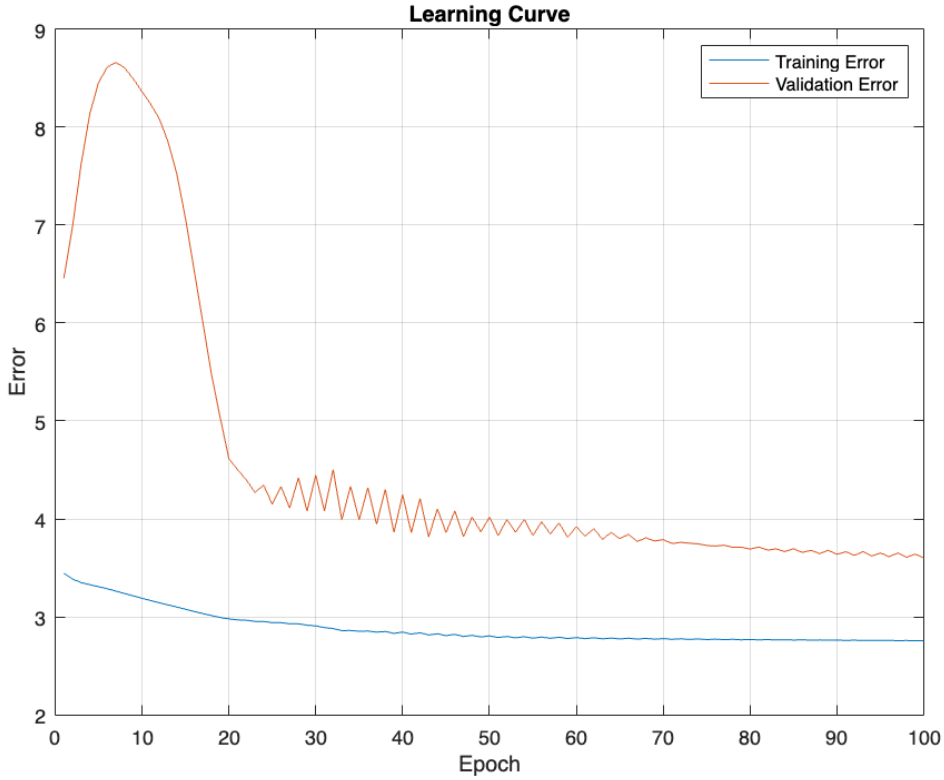
```
% Define configuration for Model 2
model_2 = struct('Part', '1','NumMFs', 3, 'InputMFT', 'gbellmf',
'OutputMFT', 'constant');

% Initialize Model 2
initial_fis_2 = initialize_tsk_model(Dtrn_1, model_2);
plot_mf(initial_fis_2, 5);
```

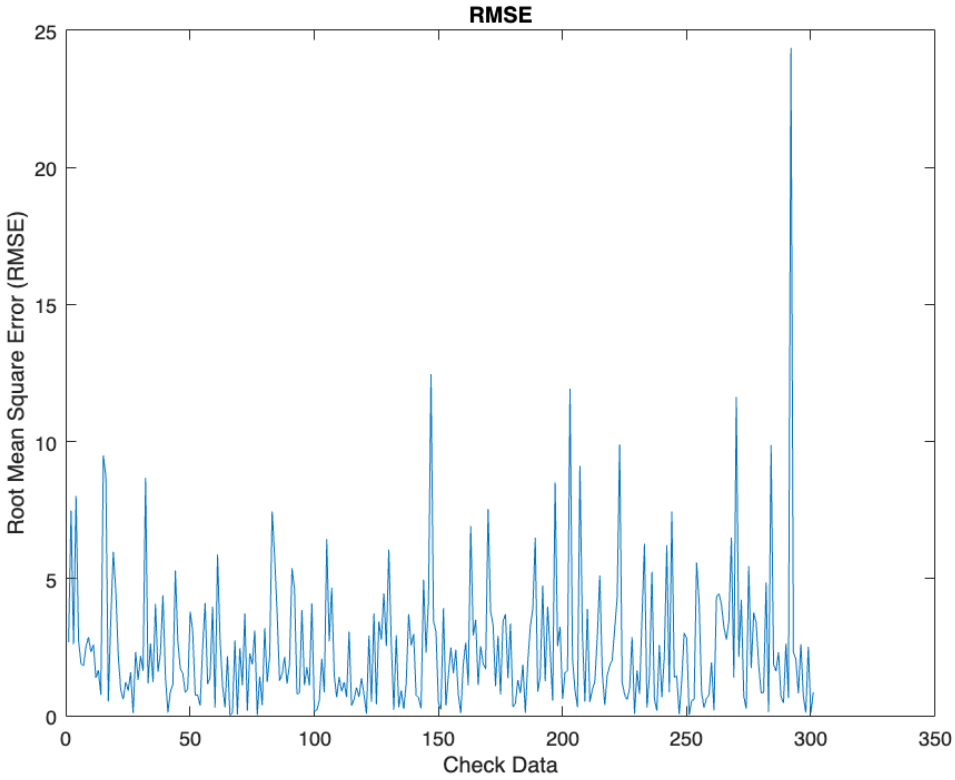


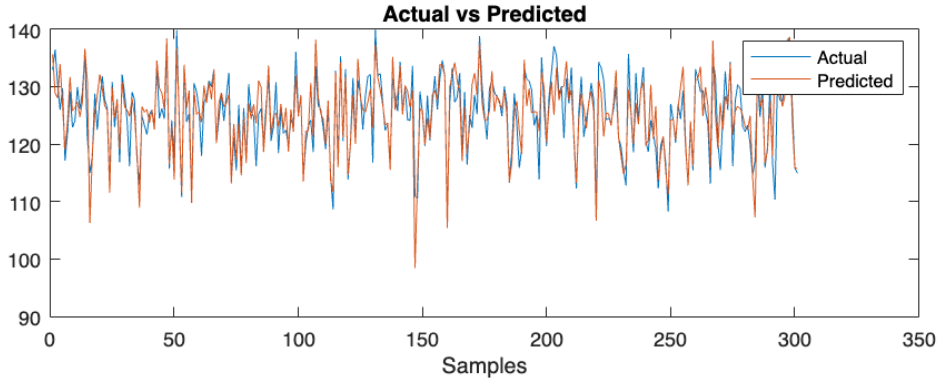

```
% Train Model 2
tic;
[trained_fis_2, training_info_2] = train_tsk_model(initial_fis_2, Dtrn_1,
Dval_1, 100);
time_2 = toc;
plot_mf(trained_fis_2, 5);
```

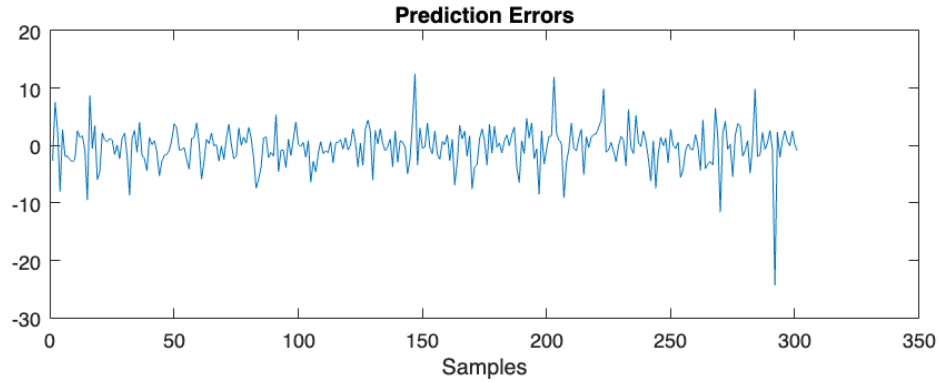



```
% Evaluate Model 2
[Rmse2, Nmse2, Ndei2, R22] = metrics_evaluation(trained_fis_2, Dchk_1);
plot_metrics(Dchk_1(:, end), evalfis(trained_fis_2, Dchk_1(:, 1:end-1)),
training_info_2.trainError, training_info_2.chkError);
```


Learning Curve
Training Error
Validation Error
Error
Epoch
9
8
7
6
5
4
3
2
0
10
20
30
40
50
60
70
80
90
100


RMSE
Root Mean Square Error (RMSE)
Check Data
25
20
15
10
5
0
0
50
100
150
200
250
300
350





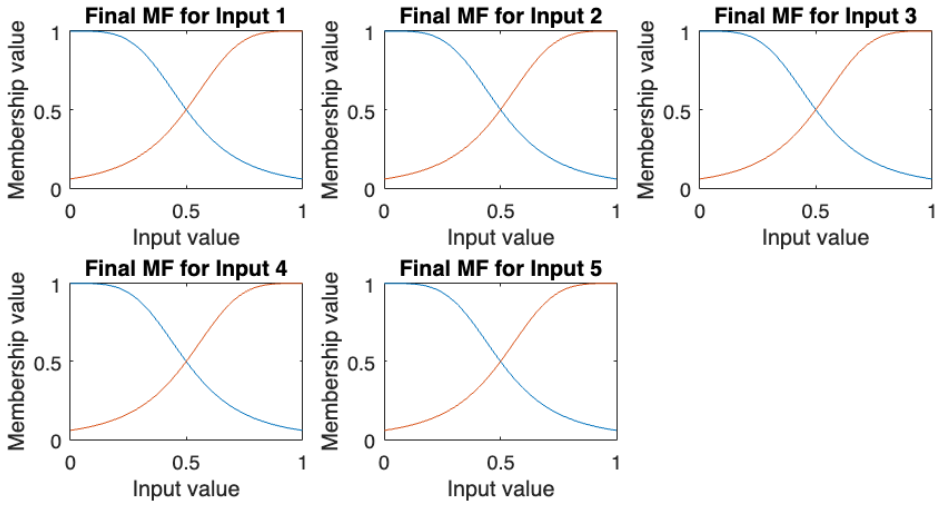

```
model_2_metrics = [Rmse2; Nmse2; Ndei2; R22];
fprintf('Error metrics for TSK model 2:\n RMSE: %.4f, NMSE: %.4f, NDEI:
%.4f, R2: %.4f\n', Rmse2, Nmse2, Ndei2, R22);
```

```
Error metrics for TSK model 2:
 RMSE: 3.5464, NMSE: 0.2882, NDEI: 0.5369, R2: 0.7118
```

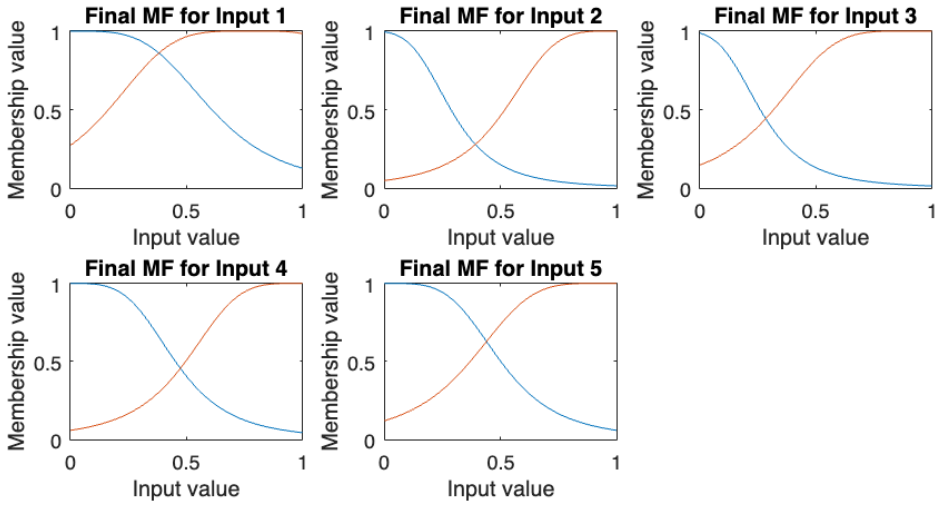
TSK Model 3

Similarly, we continue with the third model, according the the required configuration.

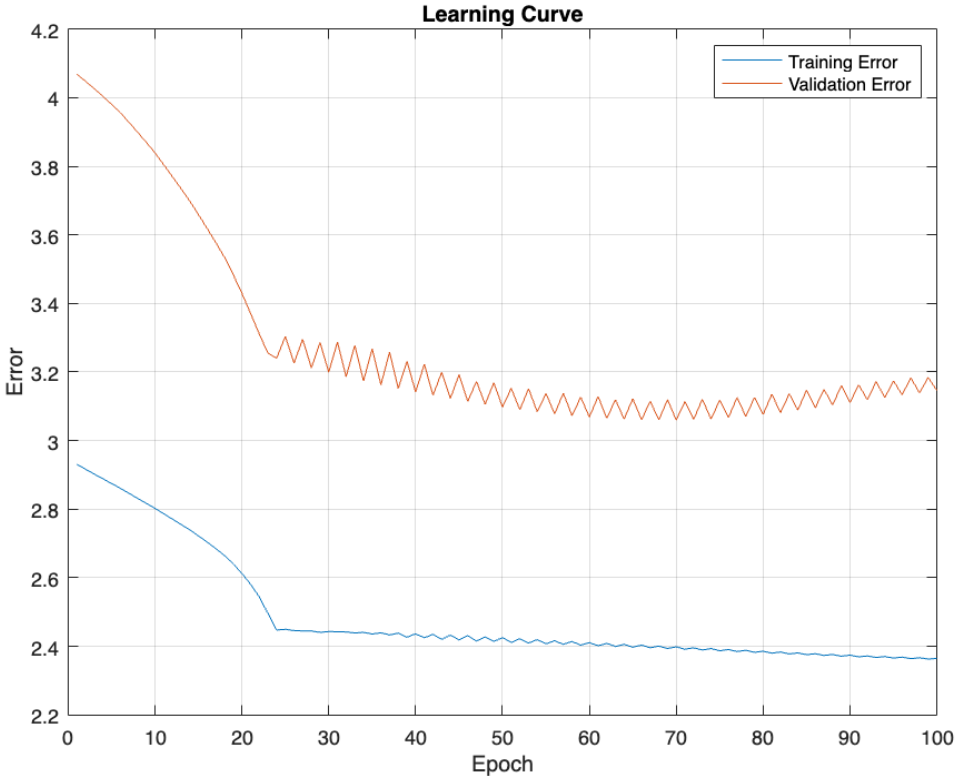
```
% Define configuration for Model 3
model_3 = struct('Part', '1','NumMFs', 2, 'InputMFT', 'gbellmf',
'OutputMFT', 'linear');

% Initialize Model 3
initial_fis_3 = initialize_tsk_model(Dtrn_1, model_3);
plot_mf(initial_fis_3, 5);
```

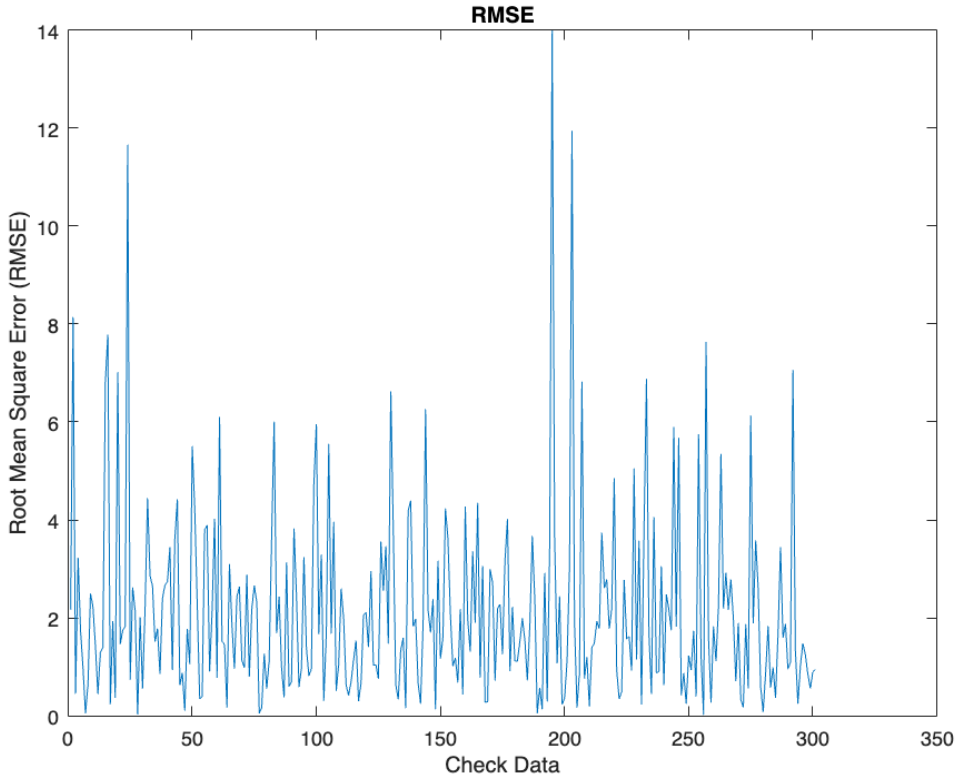


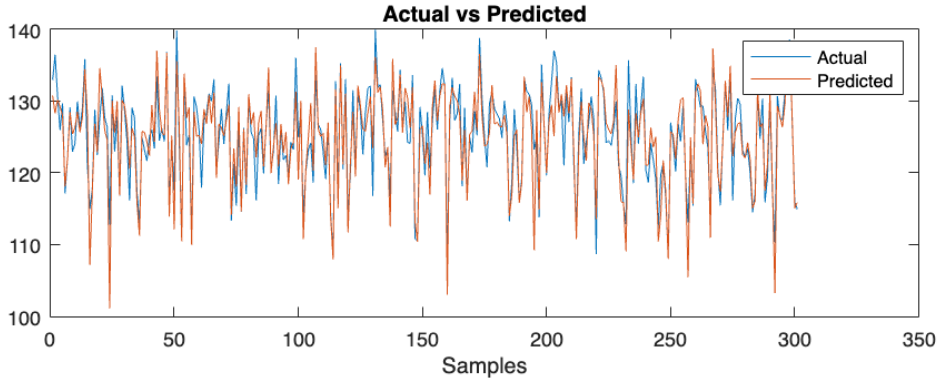

```
% Train Model 3
tic;
[trained_fis_3, training_info_3] = train_tsk_model(initial_fis_3, Dtrn_1,
Dval_1, 100);
time_3 = toc;
plot_mf(trained_fis_3, 5);
```

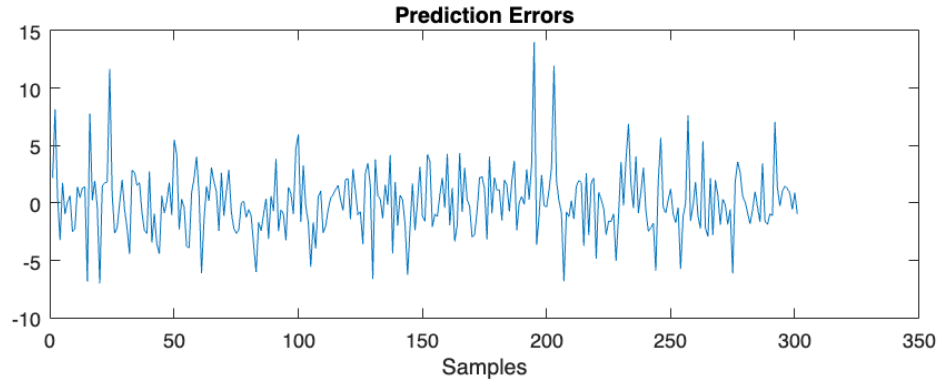



```
% Evaluate Model 3
[Rmse3, Nmse3, Ndei3, R23] = metrics_evaluation(trained_fis_3, Dchk_1);
plot_metrics(Dchk_1(:, end), evalfis(trained_fis_3, Dchk_1(:, 1:end-1)),
training_info_3.trainError, training_info_3.chkError);
```


Learning Curve
Training Error
Validation Error
Error
Epoch
4.2
4
3.8
3.6
3.4
3.2
3
2.8
2.6
2.4
2.2
0
10
20
30
40
50
60
70
80
90
100


RMSE
Root Mean Square Error (RMSE)
Check Data
14
12
10
8
6
4
2
0
0
50
100
150
200
250
300
350





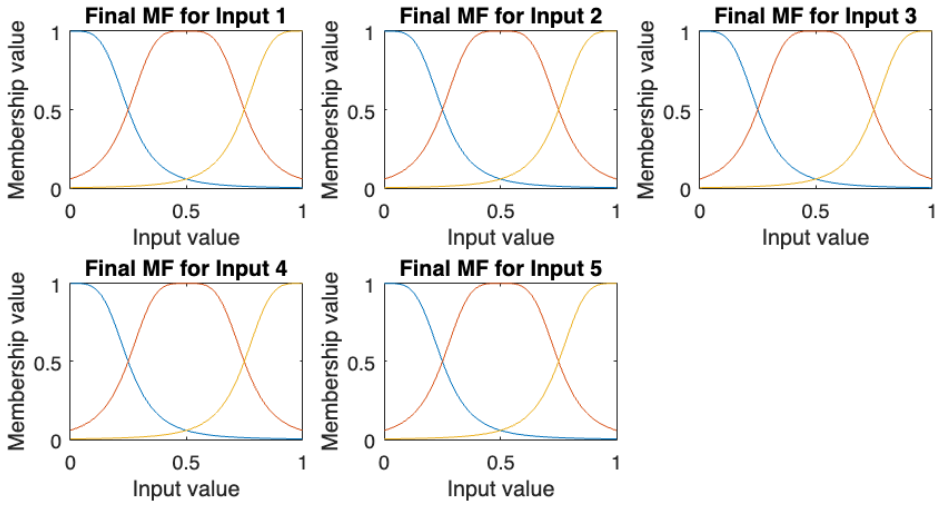

```
model_3_metrics = [Rmse3; Nmse3; Ndei3; R23];
fprintf('Error metrics for TSK model 3:\n RMSE: %.4f, NMSE: %.4f, NDEI:
%.4f, R2: %.4f\n', Rmse3, Nmse3, Ndei3, R23);
```

```
Error metrics for TSK model 3:
 RMSE: 2.8851, NMSE: 0.1908, NDEI: 0.4368, R2: 0.8092
```

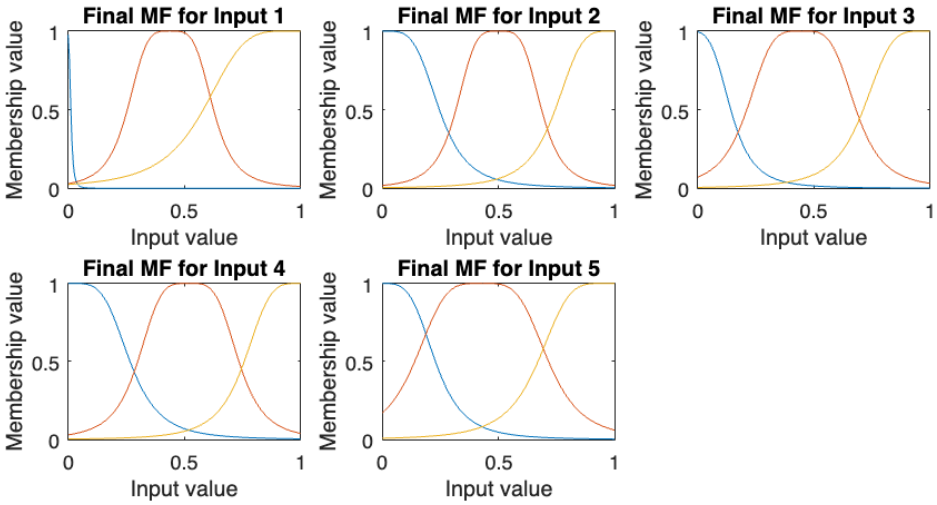
TSK Model 4

Finally, the fourth and last model trained for this dataset, is implemented below.

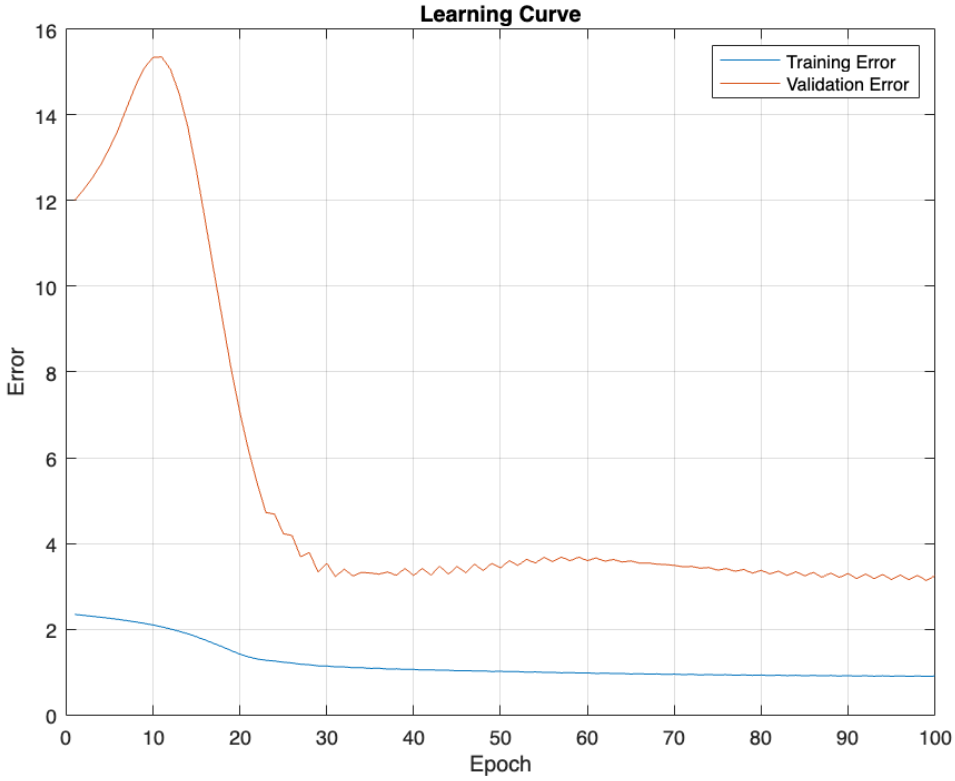
```
% Define configuration for Model 4
model_4 = struct('Part', '1','NumMFs', 3, 'InputMFT', 'gbellmf',
'OutputMFT', 'linear');

% Initialize Model 4
initial_fis_4 = initialize_tsk_model(Dtrn_1, model_4);
plot_mf(initial_fis_4, 5);
```

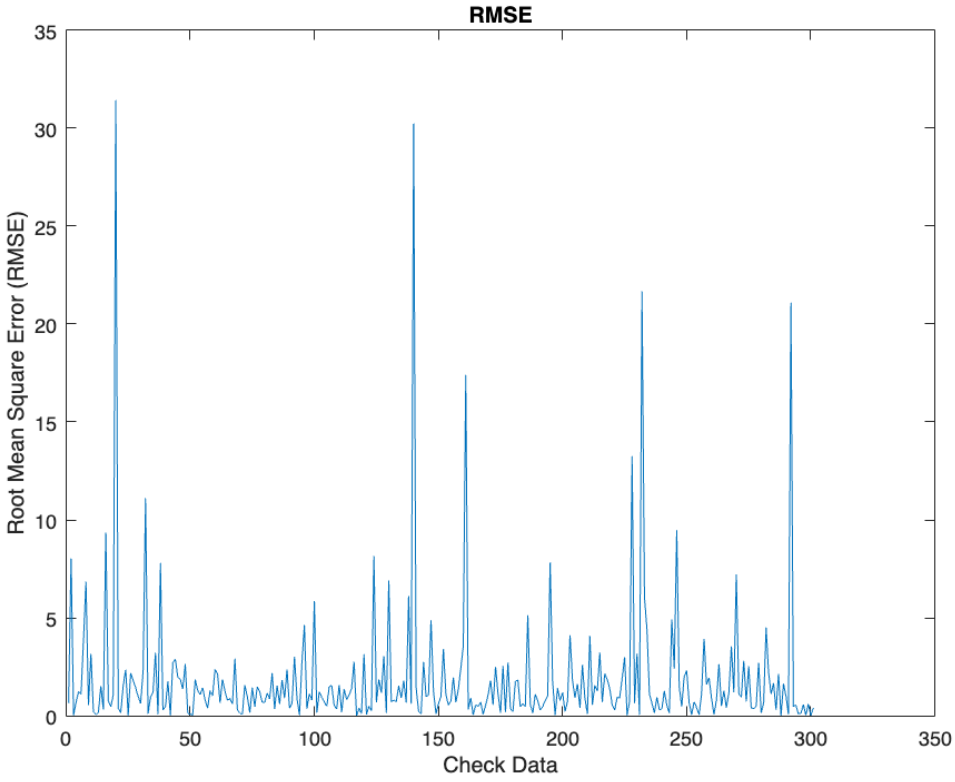


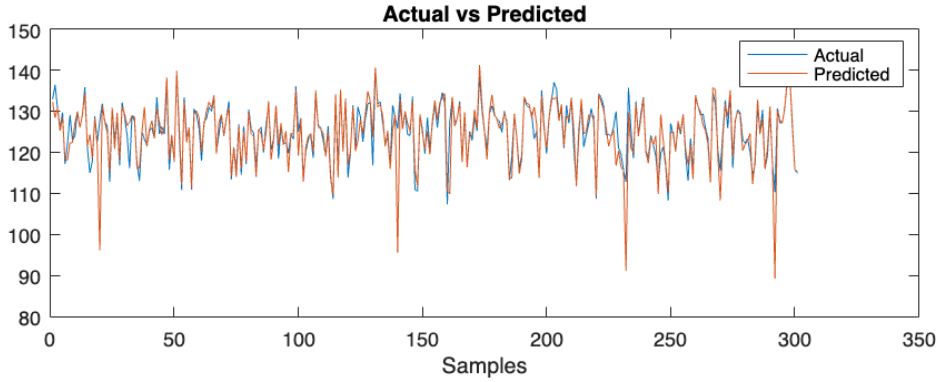

```
% Train Model 4
tic;
[trained_fis_4, training_info_4] = train_tsk_model(initial_fis_4, Dtrn_1,
Dval_1, 100);
time_4 = toc;
plot_mf(trained_fis_4, 5);
```

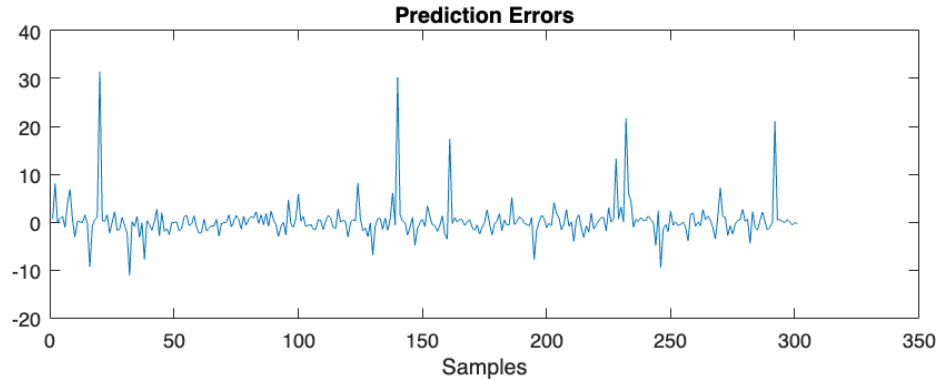



```
% Evaluate Model 4
[Rmse4, Nmse4, Ndei4, R24] = metrics_evaluation(trained_fis_4, Dchk_1);
plot_metrics(Dchk_1(:, end), evalfis(trained_fis_4, Dchk_1(:, 1:end-1)),
training_info_4.trainError, training_info_4.chkError);
```


Learning Curve
Training Error
Validation Error
Error
Epoch
0
2
4
6
8
10
12
14
16
0
10
20
30
40
50
60
70
80
90
100


RMSE
Root Mean Square Error (RMSE)
Check Data
0
5
10
15
20
25
30
35
0
50
100
150
200
250
300
350






```
model_4_metrics = [Rmse4; Nmse4; Ndei4; R24];
fprintf('Error metrics for TSK model 4:\n RMSE: %.4f, NMSE: %.4f, NDEI:
%.4f, R2: %.4f\n', Rmse4, Nmse4, Ndei4, R24);
```

```
Error metrics for TSK model 4:
 RMSE: 3.9961, NMSE: 0.3660, NDEI: 0.6050, R2: 0.6340
```

Comments and Results

Below we can see the comparison of the 4 models in regard to the evaluating metrics defined previously.

```
Metrics = {'RMSE'; 'NMSE'; 'NDEI'; 'R2'};
EvalTable = table(Metrics, model_1_metrics, model_2_metrics,
model_3_metrics, model_4_metrics);
disp(EvalTable);
```

Metrics	model_1_metrics	model_2_metrics	model_3_metrics	model_4_metrics
{'RMSE'}	3.8397	3.5464	2.8851	3.9961
{'NMSE'}	0.33789	0.28824	0.19076	0.36598
{'NDEI'}	0.58129	0.53688	0.43676	0.60496
{'R2' }	0.66211	0.71176	0.80924	0.63402

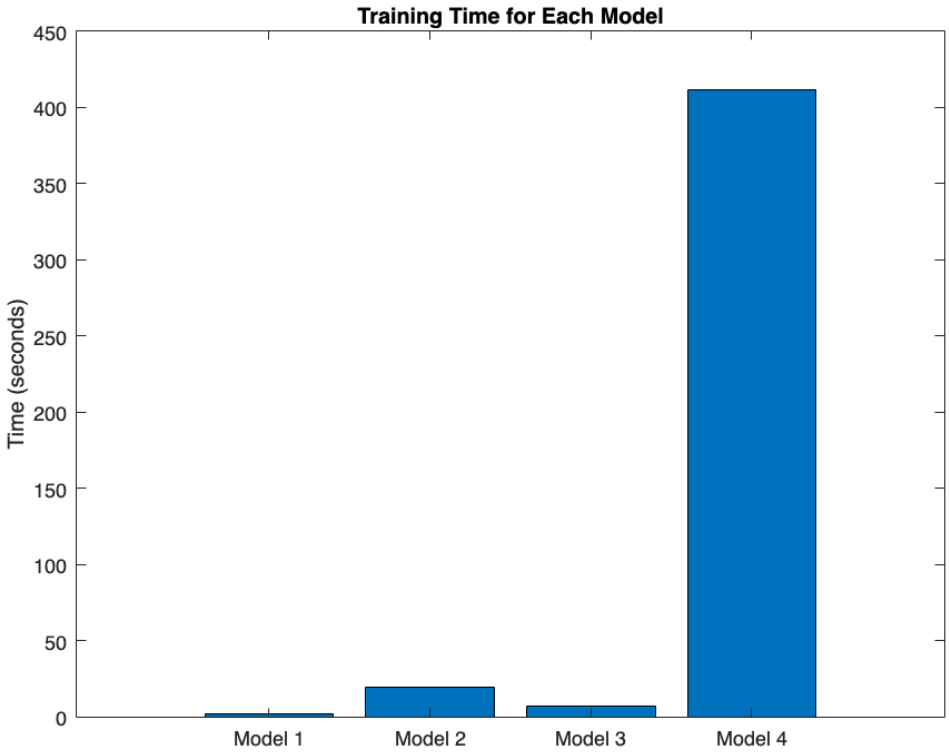
Based on the results above, we can see that the third model, significantly outperforms the rest of the models in every metric. Model 3 incorporates 2 membership functions and a polynomial output. We can see that the output type plays an important role in the model's performance. The singleton output offers more consistent results, whereas the polynomial suggests a correlation with the number of membership functions, since there is

a significant divergence in their results between 2 and 3 membership functions. So considering the above, we can say that between the 4 different combinations, the model 3's provides the best balance between complexity and efficiency.

In all four models, we do not observe conclusive signs of overfitting. Although there is an occasional trend suggesting a potential for overfit, they ultimately diverge toward stabilization of the validation error, indicating that the models generalize well to new data.

Finally, we can see the computational cost in terms of time it took for each model to be trained, below:

```
% Create a bar graph for training times
figure;
bar([time_1, time_2, time_3, time_4]);
xticklabels({'Model 1', 'Model 2', 'Model 3', 'Model 4'});
ylabel('Time (seconds)');
title('Training Time for Each Model');
```



We can clearly see the correlation between the number of membership functions and time complexity. The time difference between model 1 and 2 where their difference is the number of membership functions, is significantly higher than the difference between model 1 and 3 where their difference lies in the output (singleton vs polynomial). When the increased number of membership functions is combined with the polynomial output, which is an additional factor for complexity, we can observe the 'explosion' in complexity. Judging from this metric as well, we can verify the previous statement, that the model 3 strikes the best balance between all the factors.

TSK Application on High Dimensionality Dataset

Problem description

In this second section we are given a high-dimensional problem, the one of superconductivity. This dataset contains 21263 data points, each with 81 features, representing several material properties and conditions under which superconductivity can occur. This dataset is particularly challenging for training a model, due to this large number of features. Training such a model would be computationally intensive, and has the potential for overfitting. Thus, we are tasked with implementing feature selection and parameter optimization algorithms in order to reduce its dimensionality, and defining the number of features to be used.

Finding the Optimal parameters

First of all, we must define the parameters with which we will create our optimized model. The parameters in question are:

- Number of features
- Cluster radius

For that we will use the script *optimize_parameters* which according to the instructions, will perform feature selection using the MRMR algorithm. Following that, it will conduct a grid search over the predefined choices for different number of features and cluster radius. Finally, to make it more robust, it uses 5-fold cross validation.

We were given three choices for the feature selection algorithm. The MRMR algorithm, was chosen over the Relieff and FMI, because given the high dimensionality of our dataset, we want to select features not only relevant but also non-redundant. It seemed especially crucial in our case, because multiple features could be correlated and potentially result in overfitting, where they included together. MRMR offered this advantage by maximizing relevance whilst minimizing inter-feature correlation.

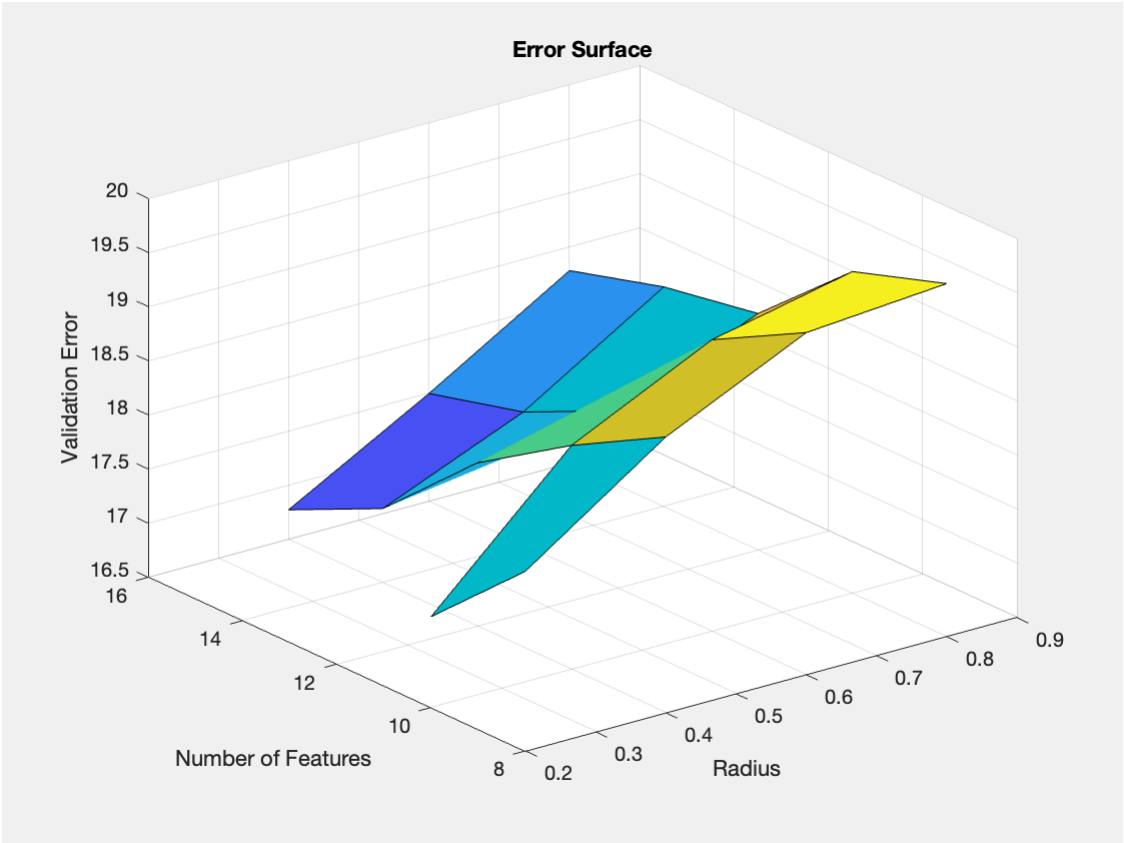
```
warning('off','all')
addpath("lib")
addpath("lib/datasets")
data_2 = table2array(readtable("datasets/superconduct.csv"));

[Dtrn_2, Dval_2, Dchk_2] = split_data(data_2);
```

```
Dimensions of Training Data:
       12758          82

Dimensions of Validation Data:
        4252          82

Dimensions of Checking Data:
        4253          82
```

We begin the optimization process, which will find the optimal parameters based on the least error, performing 100 tests on combinations:

```
% Numbers of features and radii to search over
feature_numbers = [8, 10, 12, 14, 16];
radius_a = [0.2, 0.4, 0.6, 0.8];
```

```
tic;
[opt_feature_num, opt_radius, opt_feature_idxs, eval_metrics] =
optimize_parameters(Dtrn_2, feature_numbers, radius_a);
```



Num_Features	Radius	ValidationError
8	0.2	18.158
8	0.4	19.048
8	0.6	19.66
8	0.8	19.761
10	0.2	17.338
10	0.4	18.569
10	0.6	19.193
10	0.8	19.471
12	0.2	NaN
12	0.4	18.006
12	0.6	18.136
12	0.8	18.683
14	0.2	NaN
14	0.4	17.184
14	0.6	17.724
14	0.8	18.528
16	0.2	NaN
16	0.4	16.772
16	0.6	17.495
16	0.8	18.278

```
Metrics for each combination of feature number and radius:
Features: 8
Radius: 0.20, RMSE: 18.4433, NMSE: 0.2921, NDEI: 0.5404, R2: 0.7079
Radius: 0.40, RMSE: 19.5627, NMSE: 0.3286, NDEI: 0.5732, R2: 0.6714
Radius: 0.60, RMSE: 19.8939, NMSE: 0.3398, NDEI: 0.5829, R2: 0.6602
Radius: 0.80, RMSE: 20.0751, NMSE: 0.3460, NDEI: 0.5882, R2: 0.6540
```

```
Features: 10
Radius: 0.20, RMSE: 18.2732, NMSE: 0.2869, NDEI: 0.5354, R2: 0.7131
Radius: 0.40, RMSE: 19.1216, NMSE: 0.3139, NDEI: 0.5603, R2: 0.6861
Radius: 0.60, RMSE: 19.5447, NMSE: 0.3280, NDEI: 0.5727, R2: 0.6720
Radius: 0.80, RMSE: 19.7978, NMSE: 0.3365, NDEI: 0.5801, R2: 0.6635

Features: 12
Radius: 0.20, RMSE: 17.3241, NMSE: 0.2577, NDEI: 0.5076, R2: 0.7423
Radius: 0.40, RMSE: 18.3960, NMSE: 0.2906, NDEI: 0.5390, R2: 0.7094
Radius: 0.60, RMSE: 18.7162, NMSE: 0.3008, NDEI: 0.5484, R2: 0.6992
Radius: 0.80, RMSE: 19.3821, NMSE: 0.3226, NDEI: 0.5679, R2: 0.6774

Features: 14
Radius: 0.20, RMSE: 17.3309, NMSE: 0.2579, NDEI: 0.5078, R2: 0.7421
Radius: 0.40, RMSE: 18.1756, NMSE: 0.2836, NDEI: 0.5325, R2: 0.7164
Radius: 0.60, RMSE: 18.1321, NMSE: 0.2823, NDEI: 0.5313, R2: 0.7177
Radius: 0.80, RMSE: 19.1942, NMSE: 0.3163, NDEI: 0.5624, R2: 0.6837

Features: 16
Radius: 0.20, RMSE: 17.2470, NMSE: 0.2557, NDEI: 0.5053, R2: 0.7443
Radius: 0.40, RMSE: 17.2830, NMSE: 0.2565, NDEI: 0.5064, R2: 0.7435
Radius: 0.60, RMSE: 17.8391, NMSE: 0.2732, NDEI: 0.5227, R2: 0.7268
Radius: 0.80, RMSE: 18.9363, NMSE: 0.3079, NDEI: 0.5548, R2: 0.6921
```

```
time_param = toc;
```

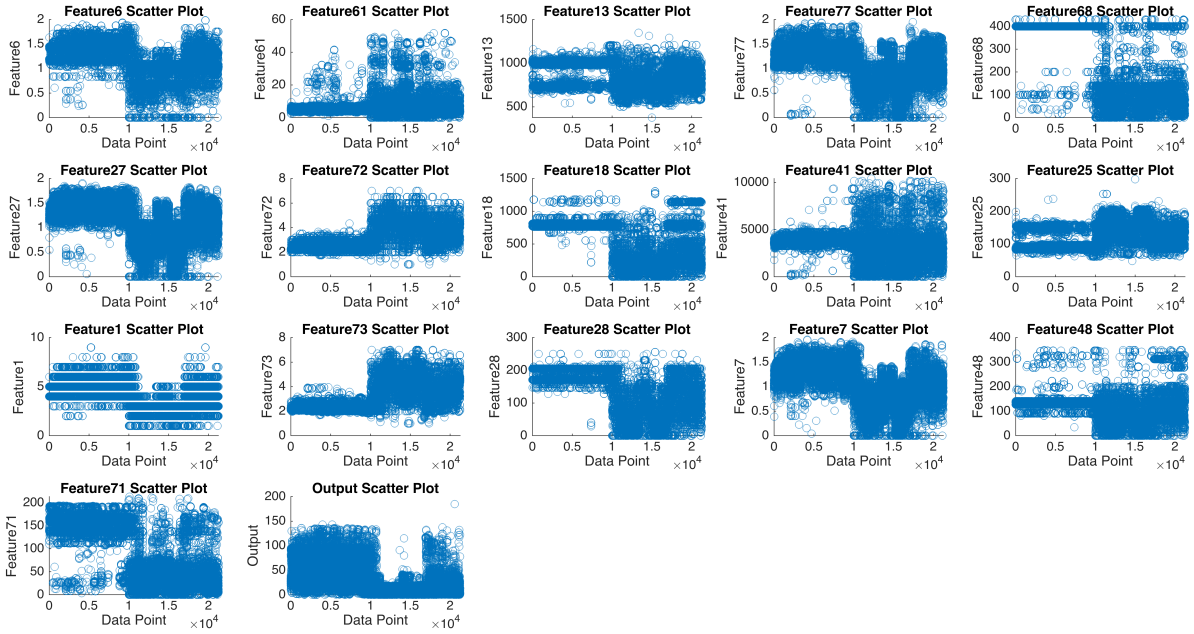
Training Optimal Model

As the process has now concluded, our optimal parameters are:

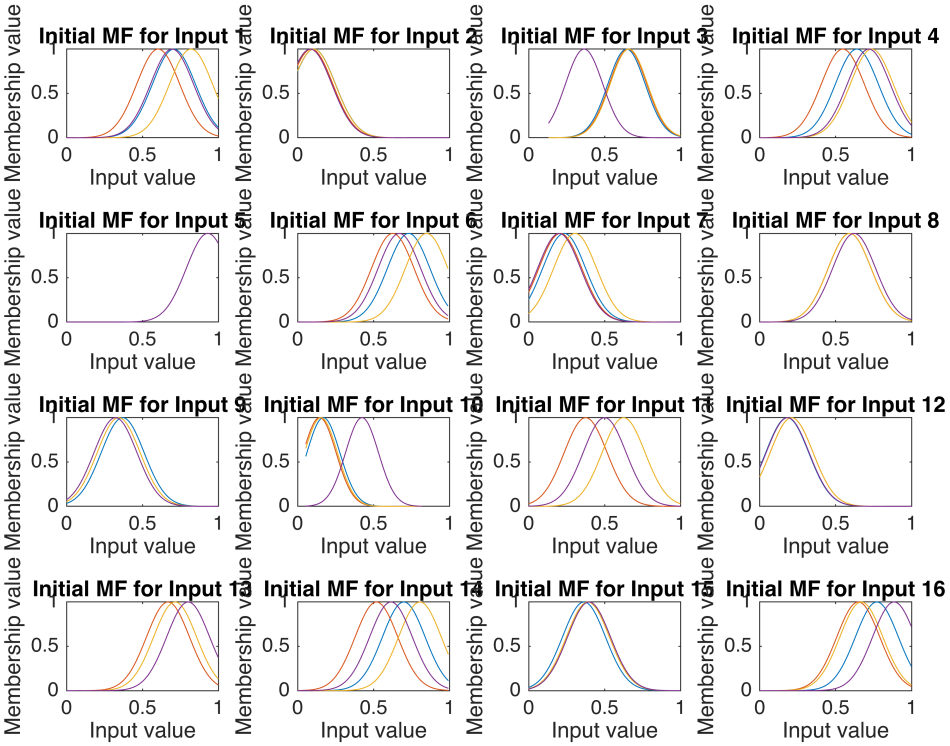
```
disp(table(opt_feature_num, opt_radius, 'VariableNames', {'Optimal Number
of Features', 'Optimal Radius'}));
```

Optimal Number of Features	Optimal Radius
16	0.4

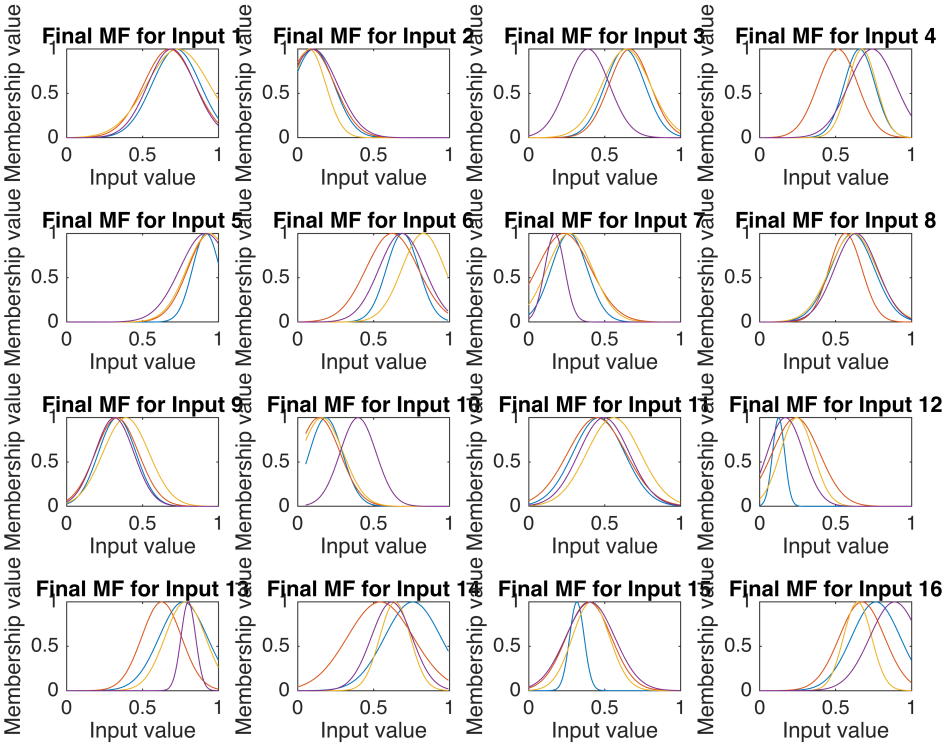
```
T = table(opt_feature_num, opt_radius, {opt_feature_idxs}, 'VariableNames',
{'Optimal_Feature_Num', 'Optimal_Radius', 'Optimal_Feature_Indices'});

% Visualize the selected features and the output
visualize_data(data_2, [], opt_feature_idxs);
```



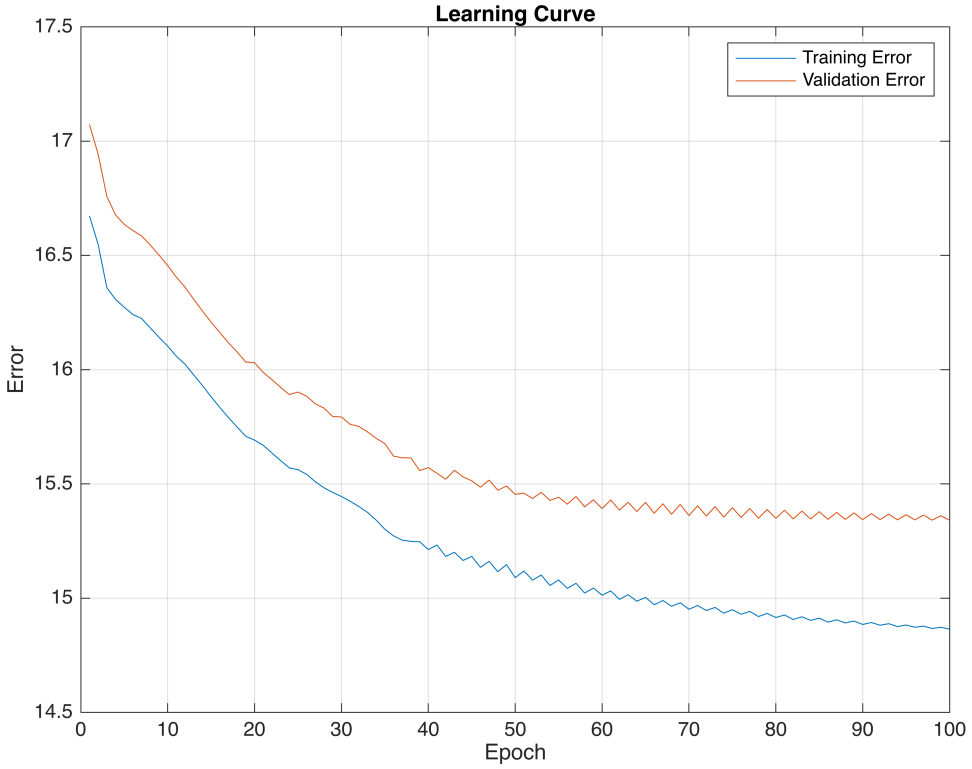

```
% Update the sets to proceed with our model training
Dtrn_opt = [Dtrn_2(:, opt_feature_idxs) Dtrn_2(:, end)];
Dval_opt = [Dval_2(:, opt_feature_idxs) Dval_2(:, end)];
Dchk_opt = [Dchk_2(:, opt_feature_idxs) Dchk_2(:, end)];

% Initialize Optimal Model
opt_model = struct('Part', '2', 'Radius', opt_radius, 'NumFeatures',
opt_feature_num);
initial_opt_fis = initialize_tsk_model(Dtrn_opt, opt_model);
plot_mf(initial_opt_fis, opt_feature_num, 1);
```

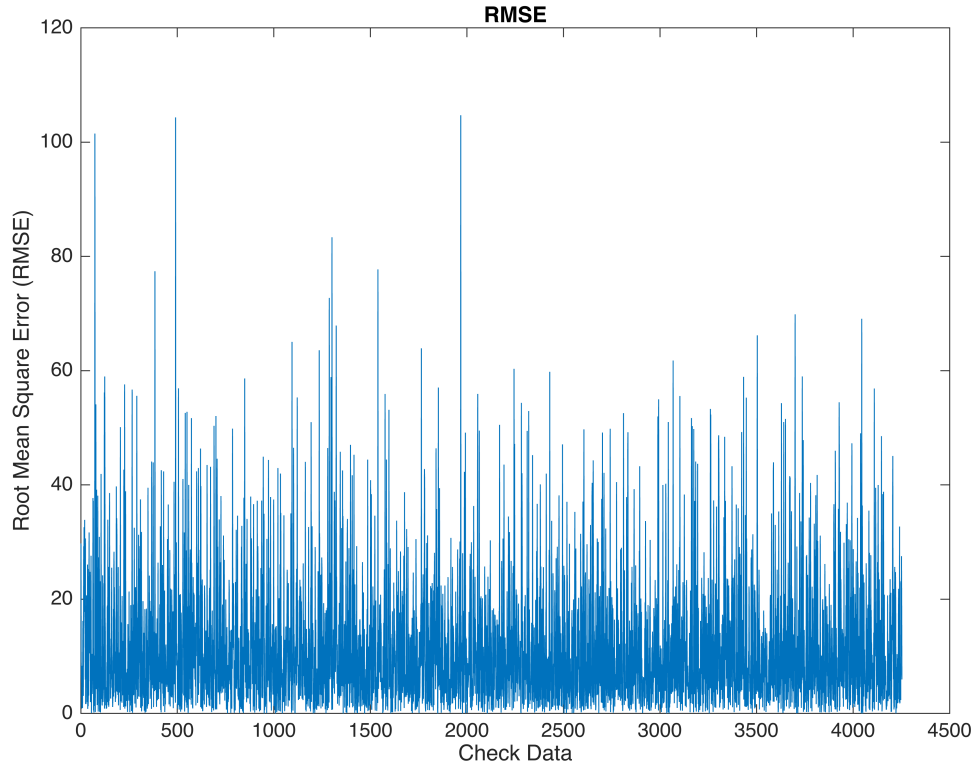


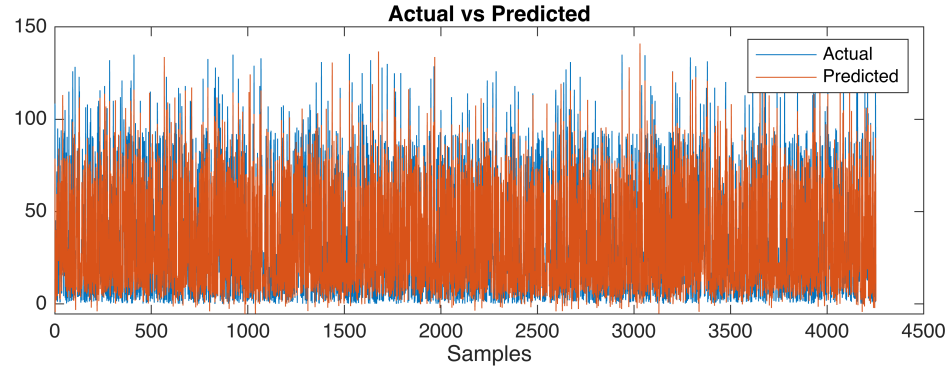

```
% Train Optimal Model
tic;
[trained_opt_fis, training_info_opt] = train_tsk_model(initial_opt_fis,
Dtrn_opt, Dval_opt, 100);
time_opt = toc;
plot_mf(trained_opt_fis, opt_feature_num, 2);
```

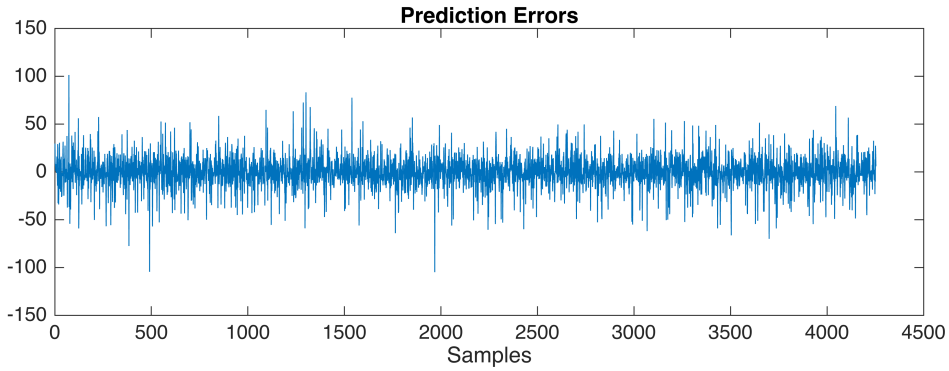



```
% Evaluate Optimal Model
[Rmse_opt, Nmse_opt, Ndei_opt, R2_opt] =
metrics_evaluation(trained_opt_fis, Dchk_opt);
plot_metrics(Dchk_opt(:, end), evalfis(trained_opt_fis, Dchk_opt(:,
1:end-1)), training_info_opt.trainError, training_info_opt.chkError);
```


Learning Curve
Training Error
Validation Error
Error
Epoch
17.5
17
16.5
16
15.5
15
14.5
0
10
20
30
40
50
60
70
80
90
100


RMSE
Root Mean Square Error (RMSE)
Check Data
120
100
80
60
40
20
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500






```
disp(table(Rmse_opt, Nmse_opt, Ndei_opt, R2_opt, 'VariableNames', {'RMSE',
'NMSE', 'NDEI', 'R2'}));
```

RMSE	NMSE	NDEI	R2
15.766	0.2127	0.4612	0.7873

Comments and Results

The optimized model, was acheived through an extensive parameter tuning and feature selection process. Utilizing the MRMR algorithm to select the most important features whilst trying to mitigate feature correlation, along with the grid search and 5 fold cross validation, offered a rubust process for optimizing the model's parameters. Given also the nature of our dataset, which comprised of 81 features, it was a necessary part of training our model, which offered a good balance between predictive effectiveness and computational performance. We also managed to avoid potential overfitting.

Our optimal model, characterized by 16 features and a radius of 0.4, proved to be a good choice, as it seems to be performing well on variance, showing a solid level of predictive accuracy. The high RMSE though, also shows signs for potential improvement.

Based on the results, we can observe a clear trend where a higher number of features used, along with a smaller radius value, generally offered better results. In the optimization process, our choice presented the lowest validation error, the second lowest RMSE and the second highest R^2.

The evaluation process was computationaly intensive, as seen below, which also showcases the rigorousness of the optimization process.

```
disp(table(time_param, time_opt, 'VariableNames', {'Parameter Optimization
Time', 'Model Training Time'}));
```

Parameter Optimization Time	Model Training Time
2135.1	77.031

Finally, we get a glimpse of the rules used in the model:

```
1×4 fisrule array with properties:

Description

Antecedent

Consequent

Weight

Connection

Details:
```

Description

```
1    "in1==in1cluster1 & in2==in2cluster1 & in3==in3cluster1 & in4==in4cluster1
& in5==in5cluster1 & in6==in6cluster1 & in7==in7cluster1 & in8==in8cluster1 &
in9==in9cluster1 & in10==in10cluster1 & in11==in11cluster1 & in12==in12cluster1 &
in13==in13cluster1 & in14==in14cluster1 & in15==in15cluster1 & in16==in16cluster1
=> out1=out1cluster1 (1)"

2    "in1==in1cluster2 & in2==in2cluster2 & in3==in3cluster2 & in4==in4cluster2
& in5==in5cluster2 & in6==in6cluster2 & in7==in7cluster2 & in8==in8cluster2 &
in9==in9cluster2 & in10==in10cluster2 & in11==in11cluster2 & in12==in12cluster2 &
in13==in13cluster2 & in14==in14cluster2 & in15==in15cluster2 & in16==in16cluster2
=> out1=out1cluster2 (1)"

3    "in1==in1cluster3 & in2==in2cluster3 & in3==in3cluster3 & in4==in4cluster3
& in5==in5cluster3 & in6==in6cluster3 & in7==in7cluster3 & in8==in8cluster3 &
in9==in9cluster3 & in10==in10cluster3 & in11==in11cluster3 & in12==in12cluster3 &
in13==in13cluster3 & in14==in14cluster3 & in15==in15cluster3 & in16==in16cluster3
=> out1=out1cluster3 (1)"

4    "in1==in1cluster4 & in2==in2cluster4 & in3==in3cluster4 & in4==in4cluster4
& in5==in5cluster4 & in6==in6cluster4 & in7==in7cluster4 & in8==in8cluster4 &
in9==in9cluster4 & in10==in10cluster4 & in11==in11cluster4 & in12==in12cluster4 &
in13==in13cluster4 & in14==in14cluster4 & in15==in15cluster4 & in16==in16cluster4
=> out1=out1cluster4 (1)"
```