Classification - TSK Models

Problem Description

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

This project is two fold. First we are given a simple dataset called 'haberman'. We then will be training 4 specific models with different configurations and evaluating their performance. Specifically, we will test the combinations below, in which we test the contribution of two parameters in the model's performance, the class dependency and the cluster's radius. We want to also see, how the number of rules is affected by each modification.

```
Models = {'Model 1'; 'Model 2'; 'Model 3'; 'Model 4'};
Class_Dependance = {'Independant'; 'Independant'; 'Dependant'; 'Dependant'};
Radius = {'0.2';'0.9'; '0.2'; '0.9'};
Combinations_Table = table(Models, Class_Dependance, Radius);
disp(Combinations_Table);
```

Models	Class_Dependance	Radius
{'Model 1'}	{'Independant'}	{'0.2'}
{'Model 2'}	{'Independant'}	{'0.9'}
{'Model 3'}	{'Dependant' }	{'0.2'}
{'Model 4'}	{'Dependant' }	{'0.9'}

The second part, includes a more complex, high-dimensional dataset '*epileptic_seizure_data*'. 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 analysed.

TSK Application on Simple Dataset

Haberman Dataset Overview

In this first section, we will utilize the 'Haberman' dataset, which consists of 306 instances, each with 3 features and one output.

Specifically:

Input:

- Age; Age of patient
- Year; Year of operation to remove nodes
- Nodes; Number of positive auxillary nodes detected

Output:

- Survival Status; Status of the person (1: lived 5 or more years, 2: died within 5 years)

```
clc; clearvars; close all;
warning('off','all')
addpath("lib")
addpath("lib/datasets")
data_1 = table2array(readtable('datasets/haberman.data', 'FileType',
'text'));

% Visualize Data
feature_names_1 = {'Age', 'Year', 'Nodes', 'Survival Status'};
visualize_data(data_1, feature_names_1, 1:3);
```

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

```
Dimensions of Training Data:
   184     4

Dimensions of Validation Data:
    61     4

Dimensions of Checking Data:
    61     4
```

TSK Model 1

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

```
% Define configuration for Model 1
model_1 = struct('Type', 'Independent', 'Radius', 0.2);

initial_fis_1 = initialize_tsk_model(Dtrn_1, model_1);
```

```
initial_fis =
  sugfis with properties:

                      Name: "sug31"
                 AndMethod: "prod"
                  OrMethod: "probor"
          ImplicationMethod: "prod"
          AggregationMethod: "sum"
      DefuzzificationMethod: "wtaver"
    DisableStructuralChecks: 0
                     Inputs: [1×3 fisvar]
                    Outputs: [1×1 fisvar]
                      Rules: [1×36 fisrule]

    See 'getTunableSettings' method for parameter optimization.
```

```
plot_mf(initial_fis_1, 3);
```

```
% 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, 3);
```

```
% Evaluate Model 1
[error_matrix, OA, PA, UA, kappa] = metrics_evaluation(trained_fis_1,
Dchk_1);
plot_metrics(error_matrix, OA, PA, UA, kappa, training_info_1.trainError,
training_info_1.chkError);
```

```
model_1_metrics = [OA; max(PA); max(UA); kappa;
length(trained_fis_1.Rules)];
```

```
fprintf('Classification metrics for the model:\n OA: %.4f, PA: %.4f, UA:
%.4f, Kappa: %.4f\n', OA, max(PA), max(UA), kappa);
```

```
Classification metrics for the model:
 OA: 0.3934, PA: 0.4651, UA: 0.8333, Kappa: 0.1068
```

TSK Model 2

We will continue implementing the second model.

```
% Define configuration for Model 2
model_2 = struct('Type', 'Independent', 'Radius', 0.9);

initial_fis_2 = initialize_tsk_model(Dtrn_1, model_2);
```

```
initial_fis =
  sugfis with properties:

                     Name: "sug31"
                AndMethod: "prod"
                 OrMethod: "probor"
          ImplicationMethod: "prod"
          AggregationMethod: "sum"
      DefuzzificationMethod: "wtaver"
    DisableStructuralChecks: 0
                     Inputs: [1×3 fisvar]
                    Outputs: [1×1 fisvar]
                      Rules: [1×2 fisrule]

    See 'getTunableSettings' method for parameter optimization.
```

```
plot_mf(initial_fis_2, 3);
```

```
% 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, 3);
```

```
% Evaluate Model 2
[error_matrix, OA, PA, UA, kappa] = metrics_evaluation(trained_fis_2,
Dchk_1);
plot_metrics(error_matrix, OA, PA, UA, kappa, training_info_2.trainError,
training_info_2.chkError);
```

```
model_2_metrics = [OA; max(PA); max(UA); kappa;
length(trained_fis_2.Rules)];
```

```
fprintf('Classification metrics for the model:\n OA: %.4f, PA: %.4f, UA:
%.4f, Kappa: %.4f\n', OA, max(PA), max(UA), kappa);
```

```
Classification metrics for the model:
 OA: 0.7049, PA: 0.8837, UA: 0.7451, Kappa: 0.1855
```

TSK Model 3

```
% Define configuration for Model 3
model_3 = struct('Type', 'Dependent', 'Radius', 0.2, 'Part', 1);

initial_fis_3 = initialize_tsk_model(Dtrn_1, model_3);
```

```
initial_fis =
  sugfis with properties:

                      Name: "fis"
                 AndMethod: "prod"
                  OrMethod: "probor"
          ImplicationMethod: "prod"
          AggregationMethod: "sum"
      DefuzzificationMethod: "wtaver"
    DisableStructuralChecks: 0
                    Inputs: [1×3 fisvar]
                   Outputs: [1×1 fisvar]
                     Rules: [1×57 fisrule]

    See 'getTunableSettings' method for parameter optimization.
```

```
plot_mf(initial_fis_3, 3);
```

```
% 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, 3);
```

```
% Evaluate Model 3
[error_matrix, OA, PA, UA, kappa] = metrics_evaluation(trained_fis_3,
Dchk_1);
plot_metrics(error_matrix, OA, PA, UA, kappa, training_info_3.trainError,
training_info_3.chkError);
```

```
model_3_metrics = [OA; max(PA); max(UA); kappa;
length(trained_fis_3.Rules)];
```

```
fprintf('Classification metrics for the model:\n OA: %.4f, PA: %.4f, UA:
%.4f, Kappa: %.4f\n', OA, max(PA), max(UA), kappa);
```

```
Classification metrics for the model:
 OA: 0.5410, PA: 0.7209, UA: 0.7045, Kappa: -0.0589
```

TSK Model 4

```
% Define configuration for Model 4
model_4 = struct('Type', 'Dependent', 'Radius', 0.9, 'Part', 1);

initial_fis_4 = initialize_tsk_model(Dtrn_1, model_4);
```

```
initial_fis =
  sugfis with properties:

                      Name: "fis"
                 AndMethod: "prod"
                  OrMethod: "probor"
          ImplicationMethod: "prod"
          AggregationMethod: "sum"
      DefuzzificationMethod: "wtaver"
    DisableStructuralChecks: 0
                    Inputs: [1×3 fisvar]
                   Outputs: [1×1 fisvar]
                     Rules: [1×2 fisrule]

    See 'getTunableSettings' method for parameter optimization.
```

```
plot_mf(initial_fis_4, 3);
```

```
% 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, 3);
```

```
% Evaluate Model 4
[error_matrix, OA, PA, UA, kappa] = metrics_evaluation(trained_fis_4,
Dchk_1);
plot_metrics(error_matrix, OA, PA, UA, kappa, training_info_4.trainError,
training_info_4.chkError);
```

```
model_4_metrics = [OA; max(PA); max(UA); kappa;
length(trained_fis_4.Rules)];
```

```
fprintf('Classification metrics for the model:\n OA: %.4f, PA: %.4f, UA:
%.4f, Kappa: %.4f\n', OA, max(PA), max(UA), kappa);
```

```
Classification metrics for the model:
 OA: 0.7213, PA: 0.9302, UA: 0.7407, Kappa: 0.1854
```

Comments and Results

Below we can see the comparison of the 4 models in regard to the evaluating metrics, as well as the number of rules

```
Metrics = {'OA'; 'PA'; 'UA'; 'Kappa'; 'No. of Rules'};
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
{'OA' }	0.39344	0.70492	0.54098	0.72131
{'PA' }	0.46512	0.88372	0.72093	0.93023
{'UA' }	0.83333	0.7451	0.70455	0.74074
{'Kappa' }	0.10685	0.18546	-0.058896	0.18539
{'No. of Rules'}	36	2	57	2

Based on the results above, we observe that model 4, which is class dependent with a radius of 0.9, offers the best performance overall. Closely following is model 2, class independent and also with a radius of 0.9. This establishes that a 'looser' radius does not necessarily lead to a drop in performance. Furthermore, class dependency doesn't significantly impact efficacy; both class dependent and independent models yield similar results with a radius of 0.9.

When we examine the role of rules, fewer seem to contribute to better performance. Model 2 and Model 4, each having only 2 rules, outperform the other models on most metrics. This challenges the conventional notion that more rules automatically yield better classification and suggests that fewer rules may improve the model's ability to generalize to unseen data.

Moving on to models with a 'tighter' radius of 0.2, performance struggles are evident. Model 1, with 36 rules, and Model 3, with 57, both exhibit compromised metrics, especially the Kappa coefficient. Intriguingly, when a small radius is paired with class dependency, as in Model 3, not only does the number of rules surge but the Kappa value also turns negative. This implies that such a model's accuracy may be largely by chance, raising concerns about its reliability in prediction.

In conclusion, in our case, maintaining a looser radius appears to be the most critical factor, enhancing generalization and overall model performance.

Examining the learning curves, we can see that they further support our findings and the models' performance. Model 1's curve shows a steady convergence, an observation consistent with its general performance. Model 2, demonstrating early stabilization, complements its position as a close second to Model 4. However, Model 3's rising validation errors echo the potential overfitting concerns, aligning with its challenging Kappa coefficient. Model 4's balanced performance between training and validation verifies its top placement in our evaluations.

In terms of rule activation and overall performance, models with a larger radius tend to do better. A larger radius likely allows for more overlap in fuzzy set projections on the inputs, making the model more flexible and effective. In contrast, a smaller radius restricts this overlap, resulting in poorer performance, as seen in Model 1 and Model 3.

In light of the data, fine-tuning the radius seems crucial for performance. A reduced rule set also shows promise for better outcomes. Balancing class dependency and compute time could further streamline model selection, demonstrating a grasp of key factors influencing model efficiency and accuracy.

Finally below, we observe the time it took for each of our models to be trained. We can see a very clear correlation between the radius, and subsequently the number of rules, with the training time, since model 1 and 3 featured significantly more rules. Interestingly though, we cannot make any observation regarding the class dependency and the training time. When comparing model 1 to 3, and model 2 to 4 (since their only difference stems from the class dependency), the former comparison seems to require more time for model 1 even though it has less rules, and in the latter, the 4th model takes more time, even though they have the same number of rules.

```
% 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');
```