Fuzzy Systems - Classification

Nikolaos Giakoumoglou 9043

03/10/2020

Abstract

The aim of this assignment is to investigate the ability of TSK models in solving classification problems. Specifically, two datasets are selected from the UCI repository for classification in their respective classes, using fuzzy neural models. The work consists of two parts, the first of which is intended for a simple investigation of the training and evaluation process of TSK models, while the second includes a more systematic approach to the problem of data learning, combined with preprocessing steps such as feature selection and model optimization methods through cross-validation.

Contents

1 Application to a Simple Dataset

1.1 Preparation (script `TSK_models_dependent.m` & `TSK_models.m`)

For this part of the assignment, we have a dataset from UCI repository called Haberman's Survival which contains 306 instances and 4 features. 7 TSK models are trained according to the following matrices, where the first 5 are class independent and the next 2 are class dependent:

	Number of Rules	Radius	Squash Factor	Output
TSK Model #1	4	0.94	0.7	Singleton
TSK Model #2	6	0.90	0.5	Singleton
TSK Model #3	8	0.76	0.5	Singleton
TSK Model #4	10	0.66	0.5	Singleton
TSK Model #5	12	0.50	0.5	Singleton

Table 1: Class Independent TSK models

	Number of Rules	Radius	Output
TSK Model #6	2	0.94	Singleton
TSK Model #7	6	0.58	Singleton

Table 2: Class Dependent TSK Models

The number of rules can be chosen by setting the radius and squash factor. The metrics we use to evaluate the models are the following:

- Overall Accuracy (OA)
- Producer's Accuracy (PA)
- User's Accuracy (UA)
- $\hat{K}$
- Error (confusion matrix)

The procedure is the following:

1. Clear workspace
2. Load Habermans dataset
3. Preprocess the dataset (`MATLAB`: `preproc`): shuffle and split into 3 non-overlapping sets: training, validation and check with equivelent proportion of output (same percentage). We can verify that with the table displayed at the beginning of the script (see Table 3). This function also normalizes the data in the unit hypercube

4. Initialize arrays for the metrics and the radius and squash factor we will use in each of the 5 models.

Then for each model

1. Generate FIS with substractive clustering options, specific radius of the cluster and squash factor (Squash factor for scaling the range of influence of cluster centers, specified as the comma-separated pair consisting of 'SquashFactor' and a positive scalar. A smaller squash factor reduces the potential for outlying points to be considered as part of a cluster, which usually creates more and smaller data clusters. This option is only for class independent FIS) as given the the arrays initialized before. Also plot the MFs. Note than in the case of class dependent TSK Models, we built the FIS from scratch with the help of `MATLAB`: `subclust` according to the example uploaded in $\mathtt{e-learning}$.

2. Set the MF of the output to constant

3. Tune the FIS (`MATLAB`: `anfis`)

4. Plot results: learning curve, prediction error and the 5 metrics using the tuned FIS for which the validation error is minimum ($\mathtt{MATLAB:chkFIS}$). Also plot the MFs after tuning

After repeating this procedure for each model, we can plot some of the metrics and compare the results.
In the following subsections we present the initial configuration of the membership functions, the membership functions after the tuning, and the metrics we mentioned before.

Class	Initial Set	Training set	Validation Set	Check Set
1	73.5294%	73.3696%	73.7705%	73.7705%
2	26.4706%	26.6304%	26.2295%	26.2295%

Table 3: Proportiong of classes for each set

Note that we trained 5 instead of 2 TSK Models for the class independent case.

1.2 TSK Model #1 (Class Independent)

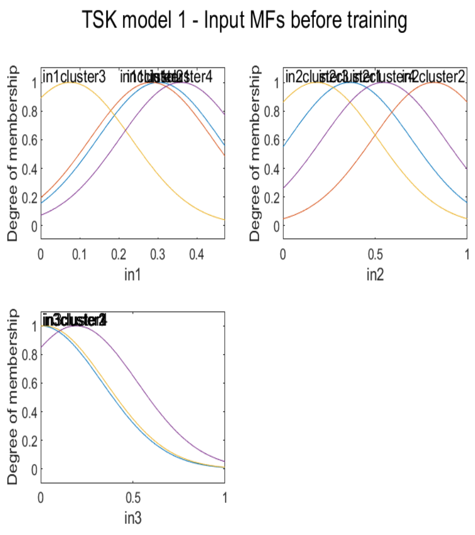


Figure 1: MF before training

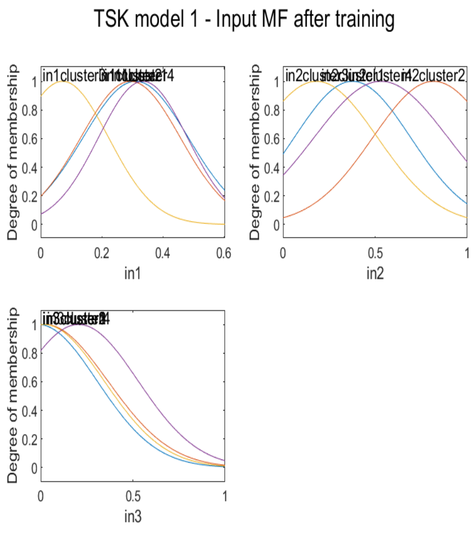


Figure 2: MF after training

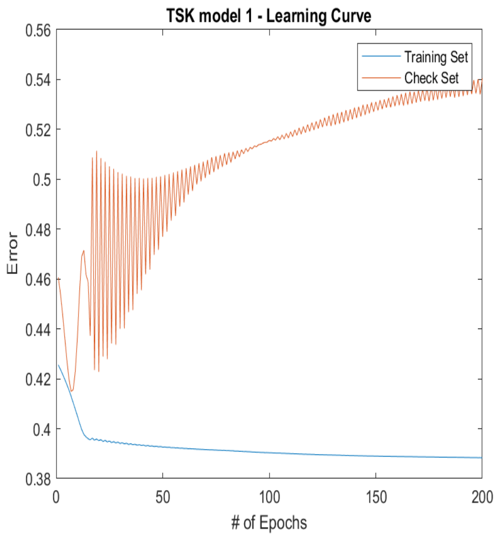


Figure 3: Learning curve (overfitting at epoch ~25)

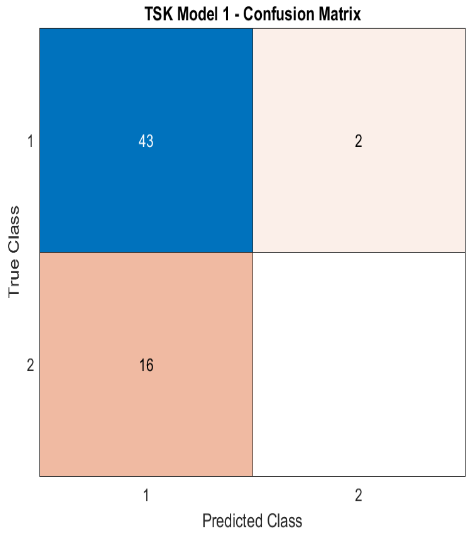


Figure 4: Confusion Matrix

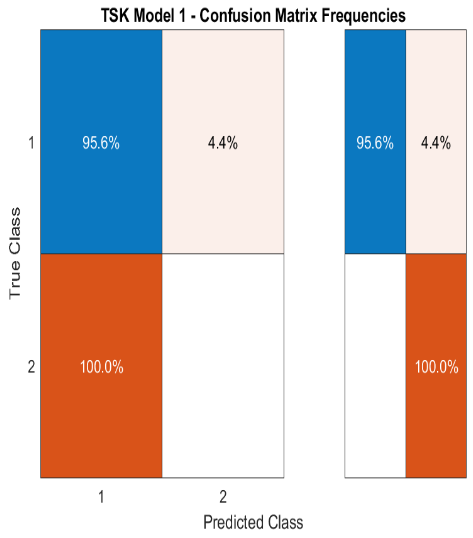


Figure 5: Confusion Matrix - Frequencies

Class Number	1	2	Value
PA	0.7288	0	-
UA	0.9555	0	-
OA	-	-	0.7049
$\hat{K}$	-	-	0.6849

Table 4: Metrics

1.3 TSK Model #2 (Class Independent)

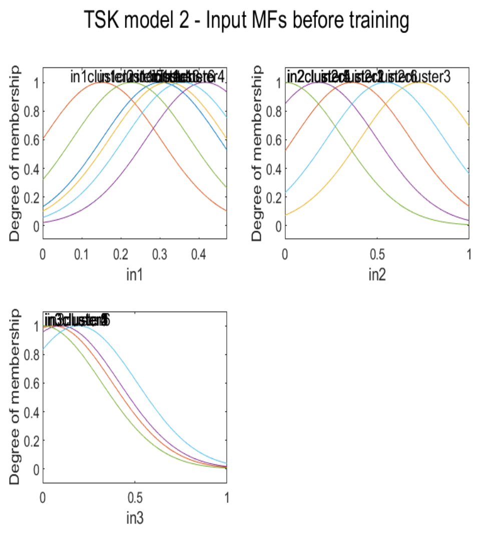


Figure 6: MF before training

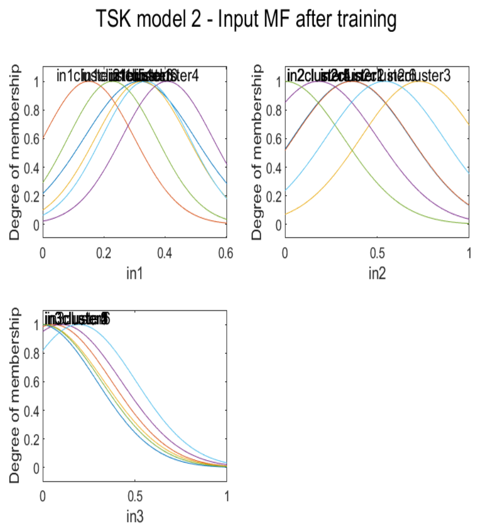


Figure 7: MF after training

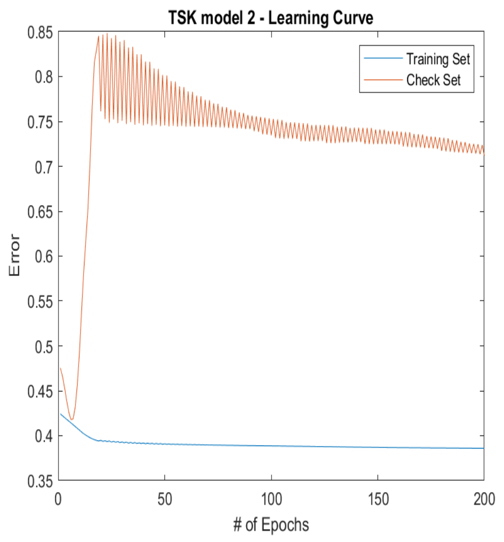


Figure 8: Learning curve (overfitting at epoch ~2)

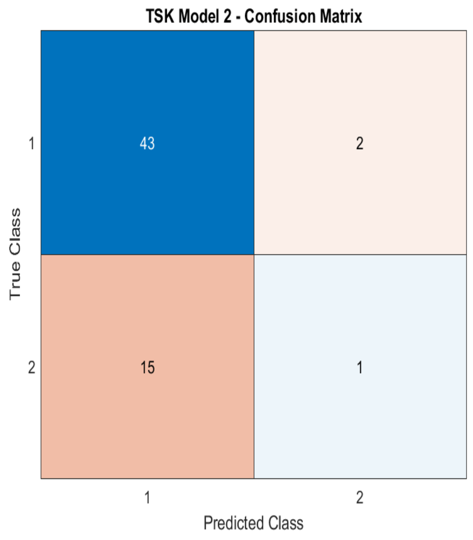


Figure 9: Confusion Matrix

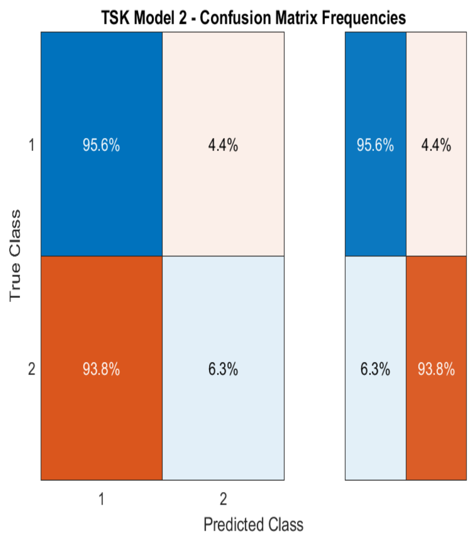


Figure 10: Confusion Matrix - Frequencies

Class Number	1	2	Value
PA	0.7414	0.333	-
UA	0.9555	0.0625	-
OA	-	-	0.7213
$\hat{K}$	-	-	0.6926

Table 5: Metrics

1.4 TSK Model #3 (Class Independent)

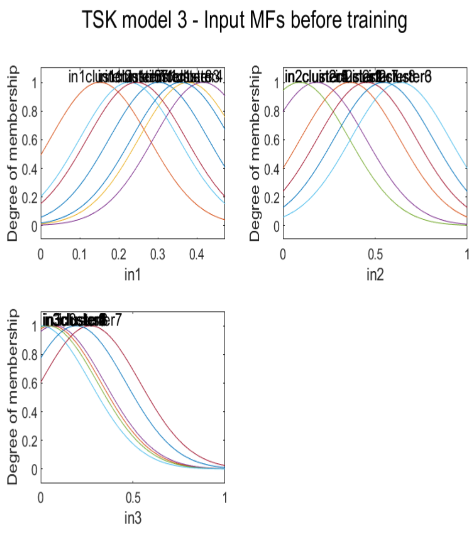


Figure 11: MF before training

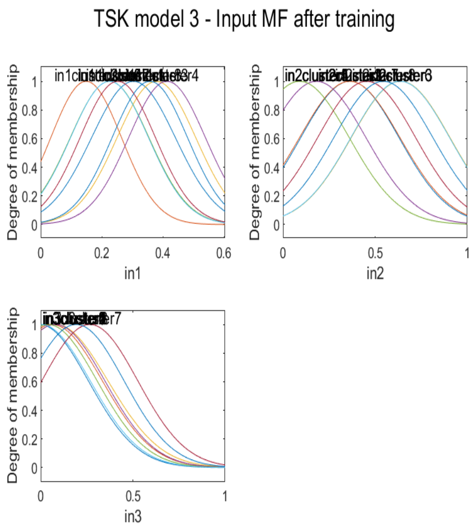


Figure 12: MF after training

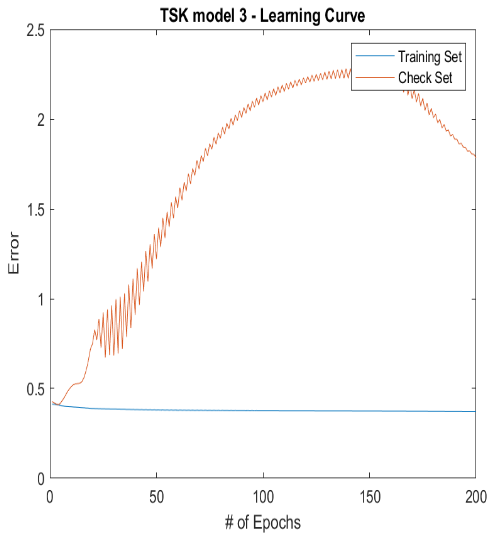


Figure 13: Learning curve (overfitting at epoch ~10)

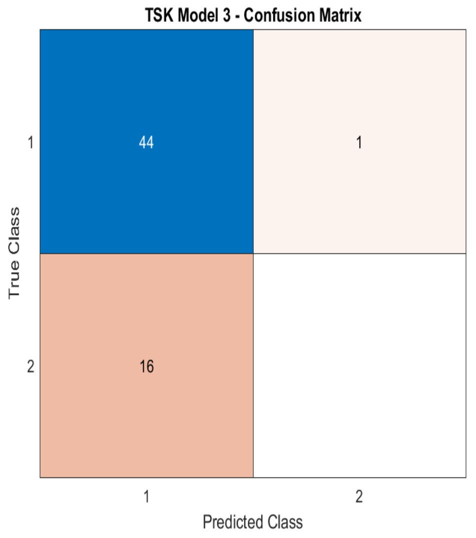


Figure 14: Confusion Matrix

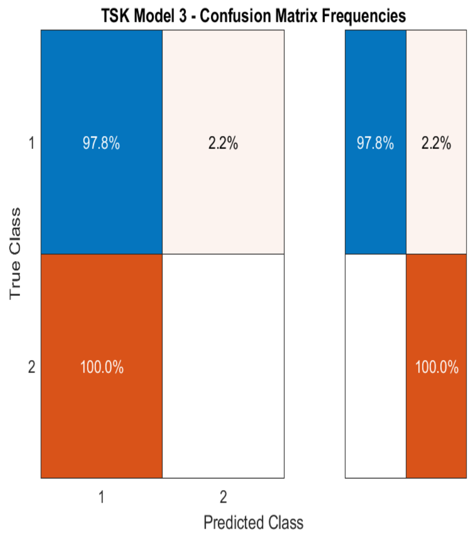


Figure 15: Confusion Matrix - Frequencies

Class Number	1	2	Value
PA	0.733	0	-
UA	0.9778	0	-
OA	-	-	0.7213
$\hat{K}$	-	-	0.7120

Table 6: Metrics

1.5 TSK Model #4 (Class Independent)

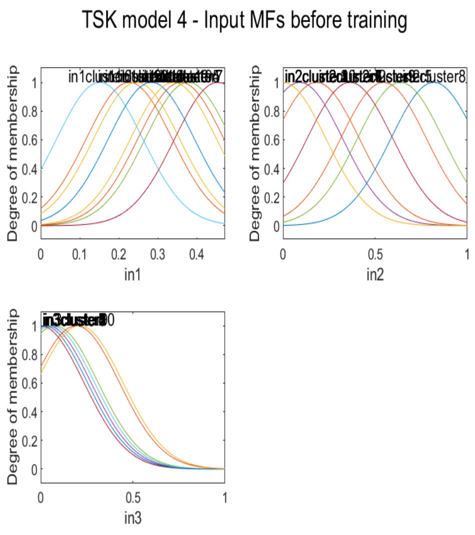


Figure 16: MF before training

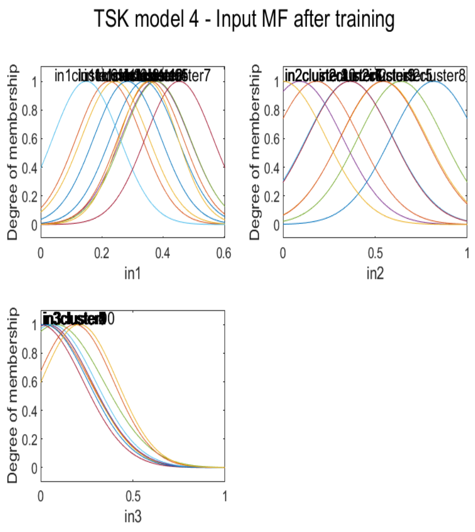


Figure 17: MF after training

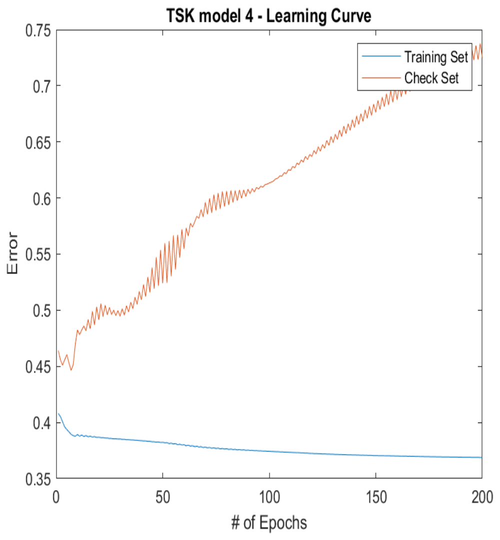


Figure 18: Learning curve (overfitting at epoch ~10)

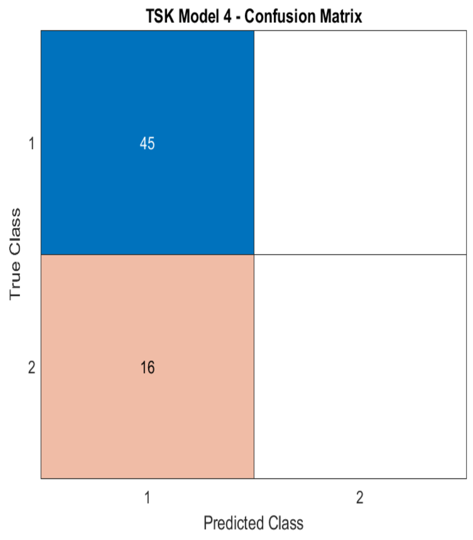


Figure 19: Confusion Matrix

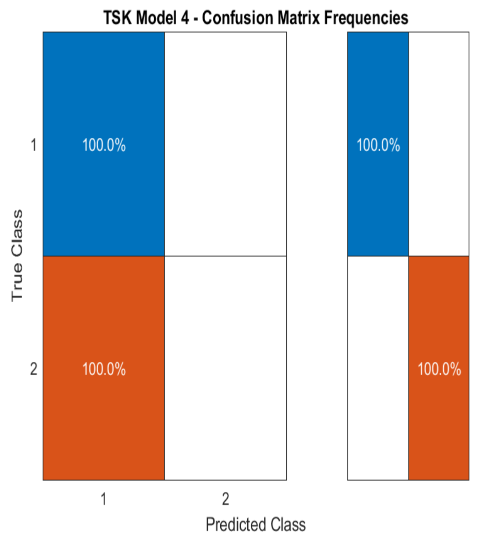


Figure 20: Confusion Matrix - Frequencies

Class Number	1	2	Value
PA	0.7333	0	-
UA	1	0	-
OA	-	-	0.7377
$\hat{K}$	-	-	0.7377

Table 7: Metrics

1.6 TSK Model #5 (Class Independent)

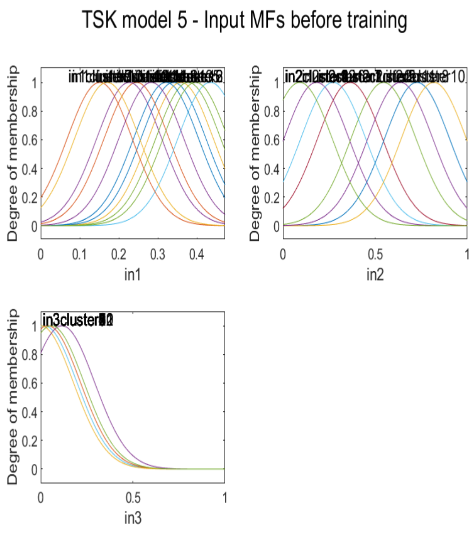


Figure 21: MF before training

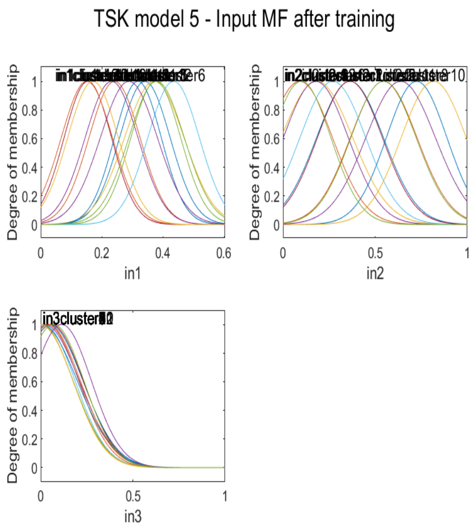


Figure 22: MF after training

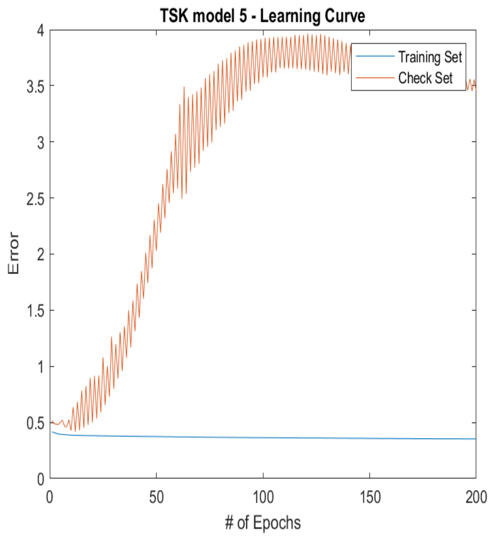


Figure 23: Learning curve (overfitting at epoch ~10)

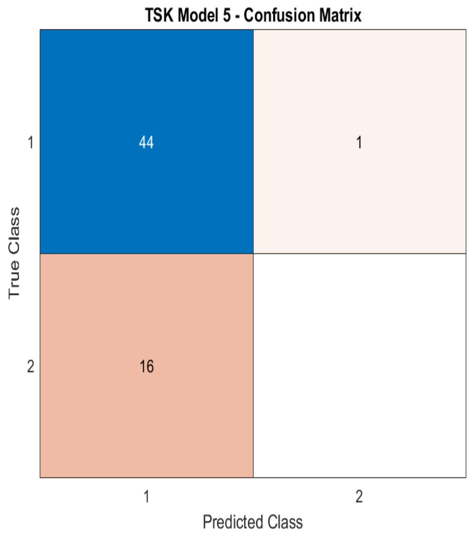


Figure 24: Confusion Matrix

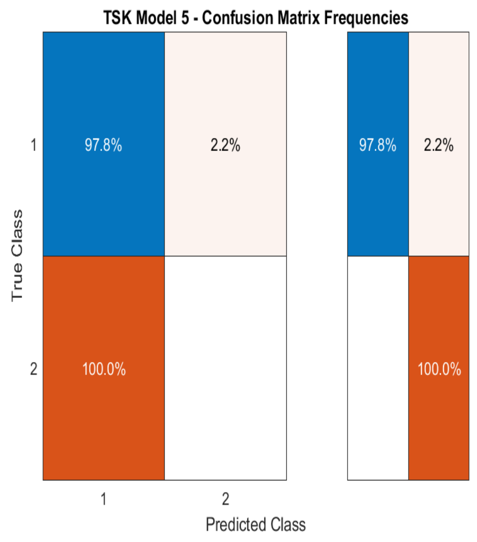


Figure 25: Confusion Matrix - Frequencies

Class Number	1	2	Value
PA	0.7333	0	-
UA	0.9778	0	-
OA	-	-	0.7213
$\hat{K}$	-	-	0.7120

Table 8: Metrics

1.7 TSK Model #6 (Class Dependent)

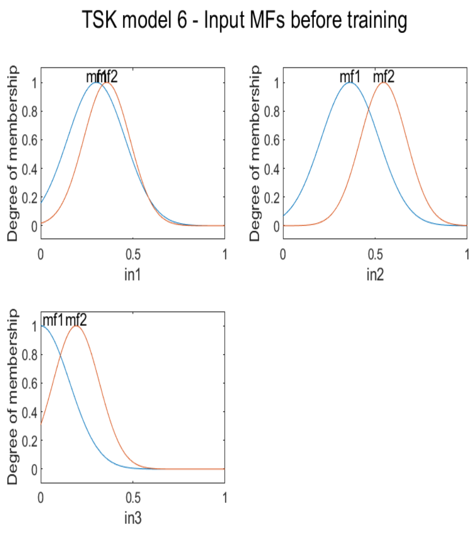


Figure 26: MF before training

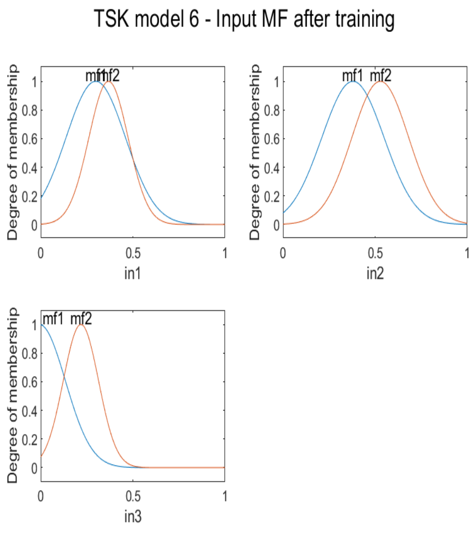


Figure 27: MF after training

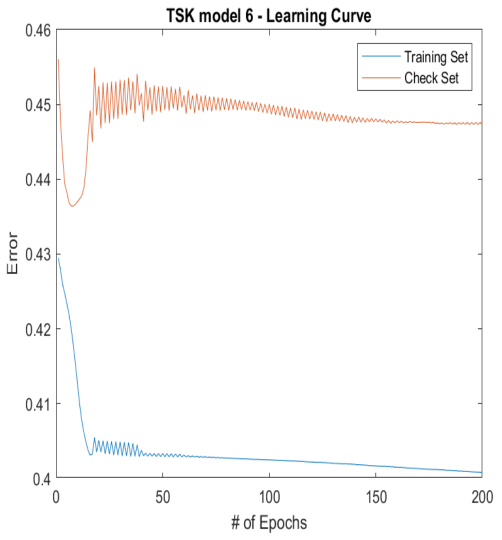


Figure 28: Learning curve (overfitting at epoch ~20)

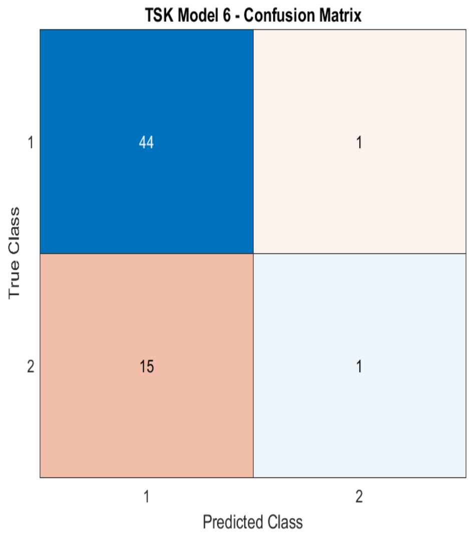


Figure 29: Confusion Matrix

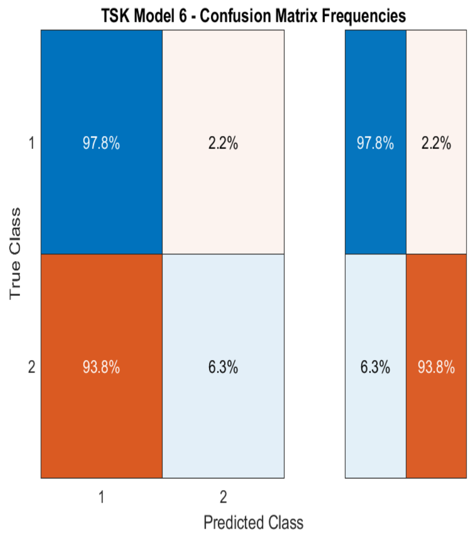


Figure 30: Confusion Matrix - Frequencies

Class Number	1	2	Value
PA	0.7458	0.500	-
UA	0.9778	0.0625	-
OA	-	-	0.7377
$\hat{K}$	-	-	0.7199

Table 9: Metrics

1.8 TSK Model #7 (Class Dependent)

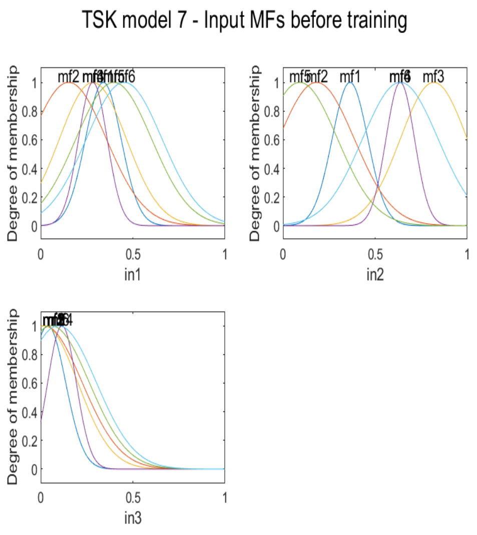


Figure 31: MF before training

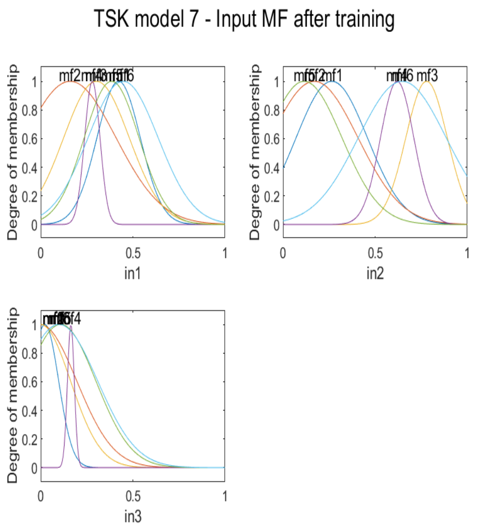


Figure 32: MF after training

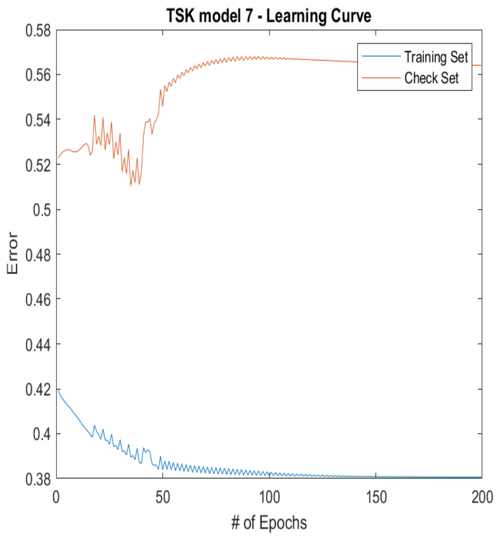


Figure 33: Learning curve (overfitting at epoch ~20)

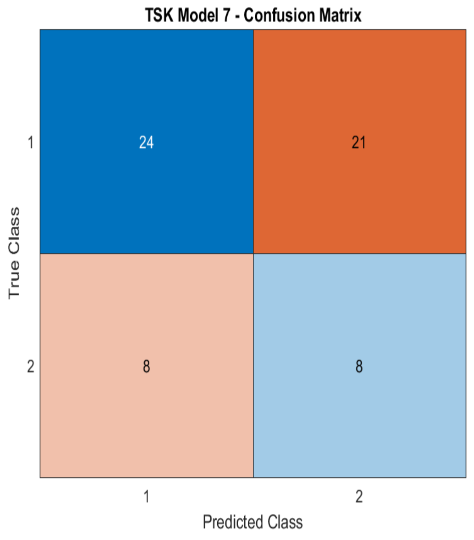


Figure 34: Confusion Matrix

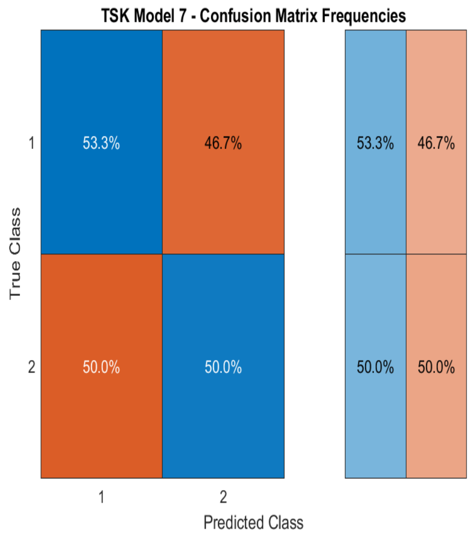


Figure 35: Confusion Matrix - Frequencies

Class Number	1	2	Value
PA	0.7500	0.2759	-
UA	0.5333	0.500	-
OA	-	-	0.5246
$\hat{K}$	-	-	0.0515

Table 10: Metrics

1.9 Conclusion

First, we present some overall results for the class independed TSK models.

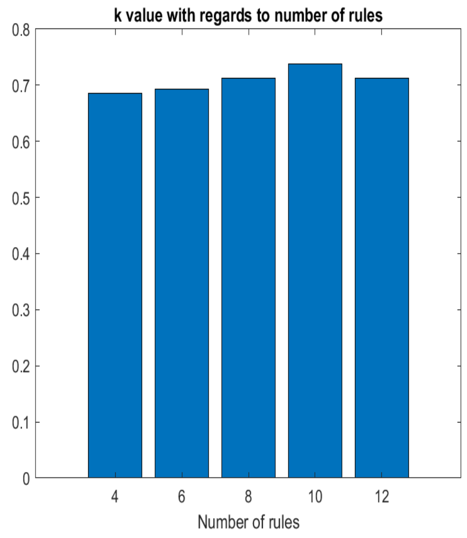


Figure 36: $\hat{K}$ vs number of rules

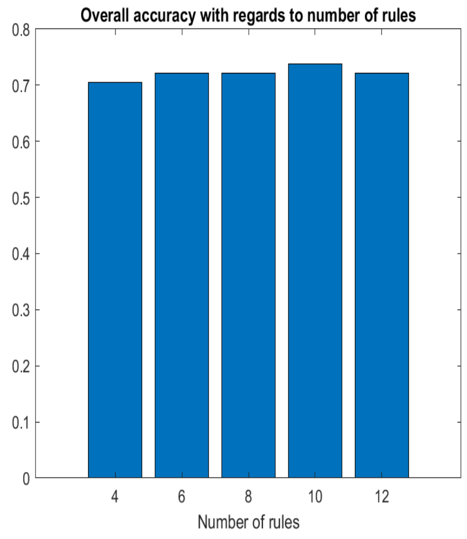


Figure 37: Overall Accuracy vs number of rules

	Number of Rules	$\hat{K}$	OA
TSK Model #1	4	0.6849	70.49 %
TSK Model #2	6	0.6926	72.13 %
TSK Model #3	8	0.7120	72.13 %
TSK Model #4	10	0.7377	73.77 %
TSK Model #5	12	0.7120	72.13 %
TSK Model #6	2	0.7199	73.77 %
TSK Model #7	6	0.0515	52.46 %

Table 11: Metrics for all 7 models

We observe that all of the models, no matter the number of rules, have similar behaviour, except for Model #7. They all predict the 1st class but fail to predict the second class, except for Model #7 which predicts both classes with 50 % accuracy. That is because we only have a few instances of the 2nd class, so the ANFIS fails to find the appropriate weights. In case we have to choose the best model, we choose model #4.

2 Application to a dataset with high dimensionality

2.1 Preparation

For this part of the assignment, we have a dataset from UCI repository called Epileptic Seizure Recognition which contains 11500 instances and 179 features. Applying a conventional TSK model is quite difficult due to the high number of rules that occur. We will use a different approach with the grid search algorithm. The procedure is the following

1. Clear workspace
2. Load Epileptic Seizure Recognition dataset
3. Preprocess the dataset (`MATLAB`: `preproc`): shuffle and split into 3 non-overlapping sets: training, validation and check with equivelent proportion of output (same percentage). We can verify that with the table displayed at the beginning of the script (see Table 8). This function also normalizes the data in the unit hypercube
4. Feature selection with 100 nearest neighbours using Relief algorithm (`MATLAB`: `relieff`) for the shuffled data
5. Arbitrarily define values for the optimal number of features and radius of cluster center's range of influence in each of the data dimensions, assuming the data falls within a unit hyperbox
6. For each pair of feature and radius (the two variables form a grid which we search for the optimal model) we generate the FIS (`MATLAB`: `genfis2` or build from scratch), perform 5-Fold Cross Validation (`MATLAB` : `cvpartition`: by default training set is 80% and check set is 20% as given by the indexes) by tuning the FIS (`MATLAB` : `anfis`) for each fold. Each fold represents a secondary model which we use to find an error using the validation set. The mean of all 5 errors of the folds represent the error of the initial model (specific value for rules and features). This procedure is called Grid Search
7. Plot the errors of the models in 2D and 3D
8. Find the optimal model; the one with the minimum MSE

Class	Initial Set	Training set	Validation Set	Check Set
1	20%	20%	20%	20%
2	20%	20%	20%	20%
3	20%	20%	20%	20%
4	20%	20%	20%	20%
5	20%	20%	20%	20%

Table 12: Proportiong of classes for each set

2.2 Find Optimal Model using Grid Search as Independent (script `grid_search.m`)

In this case we use $3, 9, 15, 21$ features with radius from the set $[0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75, 0.85, 0.95]$, Each cell is based on the following table

Features	Radius								
3 Features	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
9 Features	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
15 Features	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
21 Features	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95

Table 13: Values of features and radiuses used

We will now present the results of the rule grid that occured and the MSE in 3 different forms (table, 2D and 3D figures).

Features	Radius Set								
3 Features	5	5	5	3	3	3	2	1	1
9 Features	4	4	4	3	3	3	2	1	1
15 Features	4	4	4	3	3	3	2	1	1
21 Features	4	4	4	3	3	2	2	1	1

Table 14: Rule Grid

Features	Radius Set								
3 Features	1.3941	1.4190	1.4184	1.4201	1.4196	1.4912	1.4156	–	–
9 Features	1.3067	1.3618	1.3648	1.3637	1.3749	1.4065	1.3834	–	–
15 Features	1.2756	1.3360	1.3545	1.3533	1.3538	1.3544	1.3633	–	–
21 Features	1.2171	1.2898	1.3074	1.3123	1.3111	1.3080	1.3129	–	–

Table 15: MSE in table form

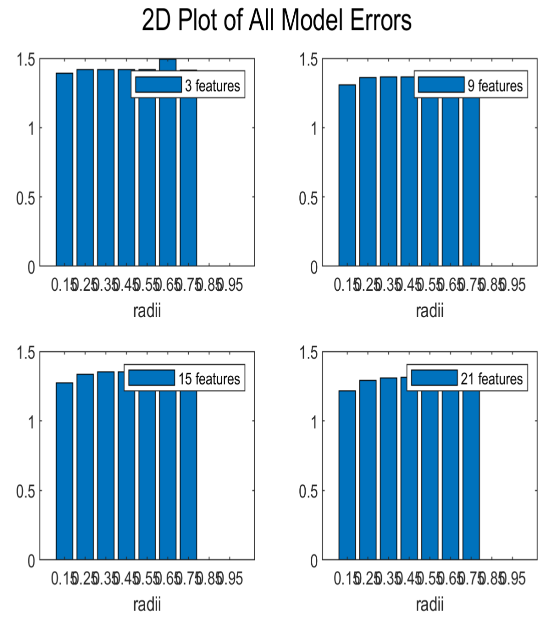


Figure 38: MSE in 2D

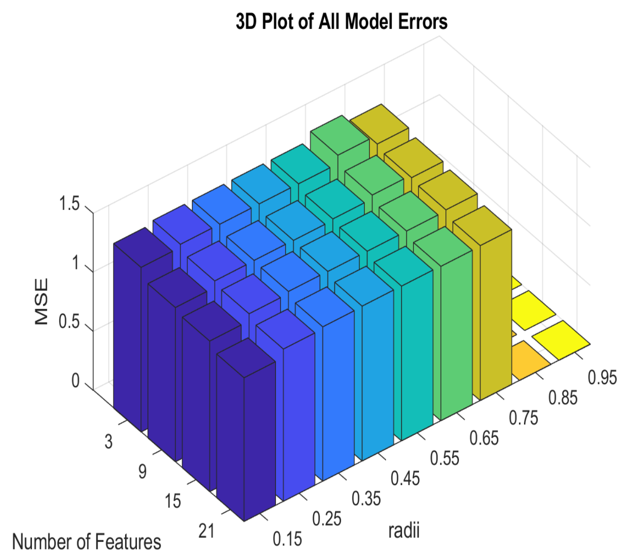


Figure 39: MSE in 3D

The minimum MSE is 1.2171 **for 21 features and radius 0.15**, which we save in `opt_model.mat`. We observe that no matter the radius and number of features we use, the MSE stays the same. Small improvement exists for higher number of features. Lower radius which means higher number of rules seems to decrease the MSE.

2.3 Optimal Model as Independent (script `opt_model.m`)

The optimal model is the one with

- MSE equals 1.2171
- 21 features
- cluster's radius 0.15
- 4 rules

We load the model which we already saved in `opt_model.mat` and train the ANFIS. Results are the following

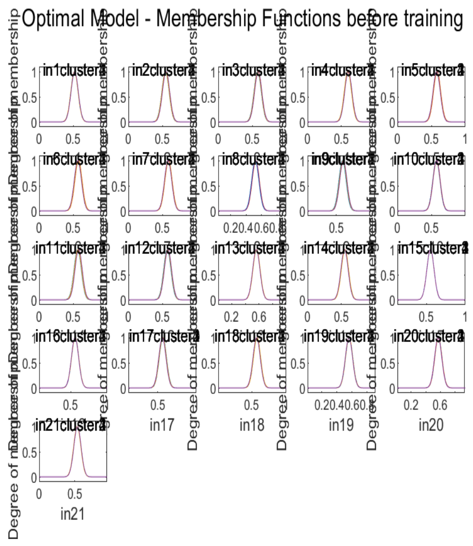


Figure 40: MF before training

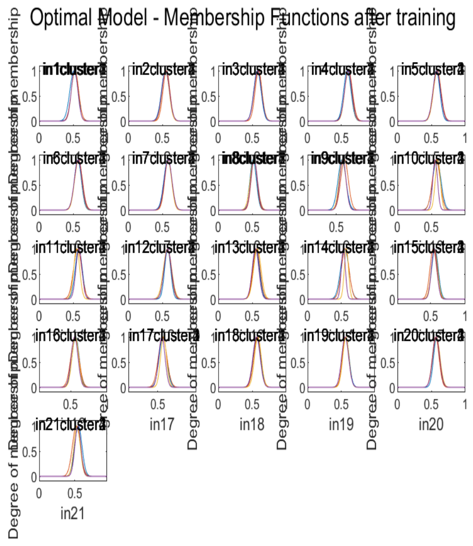


Figure 41: MF after training

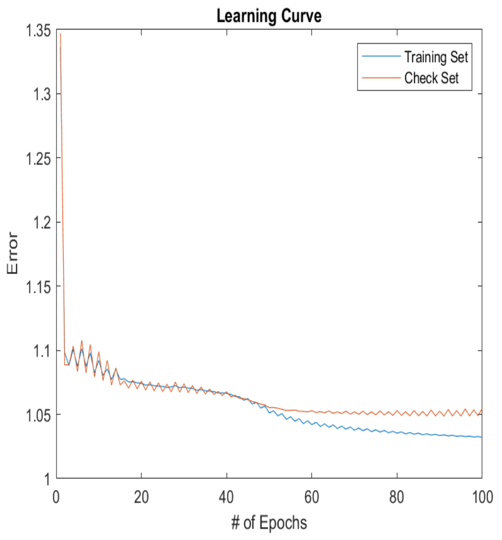
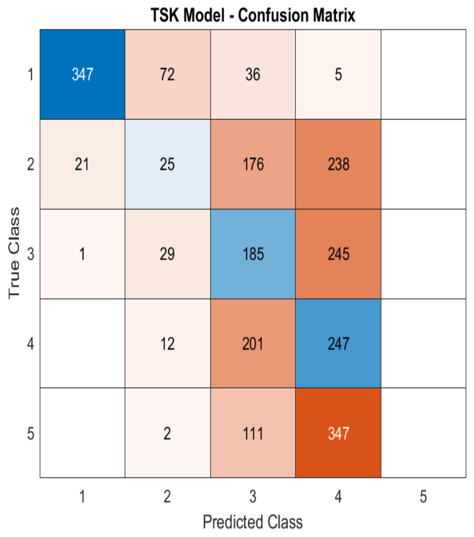


Figure 42: Learning curve (overfitting at epoch ~50)



	1	2	3	4	5
1	347	72	36	5	
2	21	25	176	238	
3	1	29	185	245	
4		12	201	247	
5		2	111	347	

Figure 43: Confusion Matrix

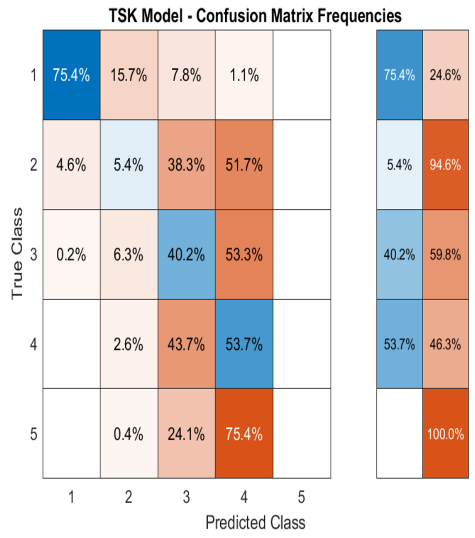


Figure 44: Confusion Matrix - Frequencies

Class Number	1	2	3	4	5	Value
PA	0.9404	0.1786	0.2609	0.2283	NaN	–
UA	0.7543	0.0543	0.4022	0.5370	0	–
OA	–	–	–	–	–	0.3496
$\hat{K}$	–	–	–	–	–	0.2224

Table 16: Metrics

2.4 Find Optimal Model using Grid Search as Dependent (script `grid_search_dependent.m`)

In this case we use $3, 9, 15, 21$ features with radius from the set $[0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75, 0.85, 0.95]$, Each cell is based on the following table

Features	Radius								
3 Features	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
9 Features	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
15 Features	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
21 Features	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95

Table 17: Values of features and radiuses used

We will now present the the MSE in 3 different forms (table, 2D and 3D figures).

Features	Radius Set								
3 Features	1.3576	1.3510	1.3667	1.3740	1.3723	1.3841	1.3746	1.3704	1.3741
9 Features	–	1.1480	1.1374	1.1280	1.1357	1.1024	1.1957	1.2454	1.2430
15 Features	–	1.1354	1.1084	1.1367	1.1058	1.1092	1.1290	1.1464	1.2279
21 Features	–	–	1.0933	1.0797	1.0943	1.0709	1.0666	1.0624	1.1537

Table 18: MSE in table form

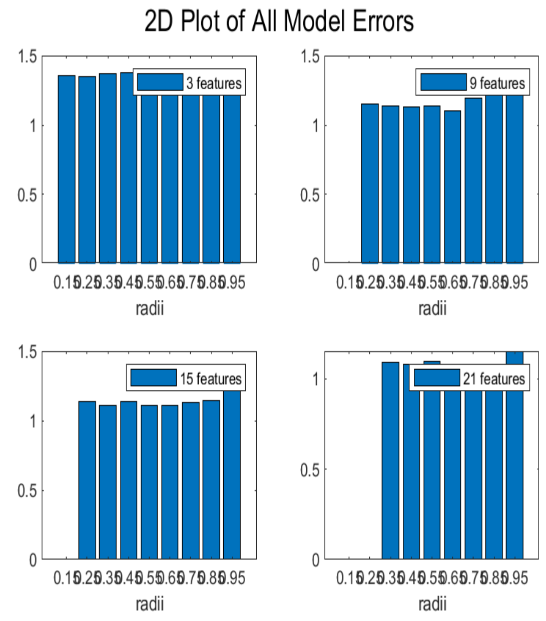


Figure 45: MSE in 2D

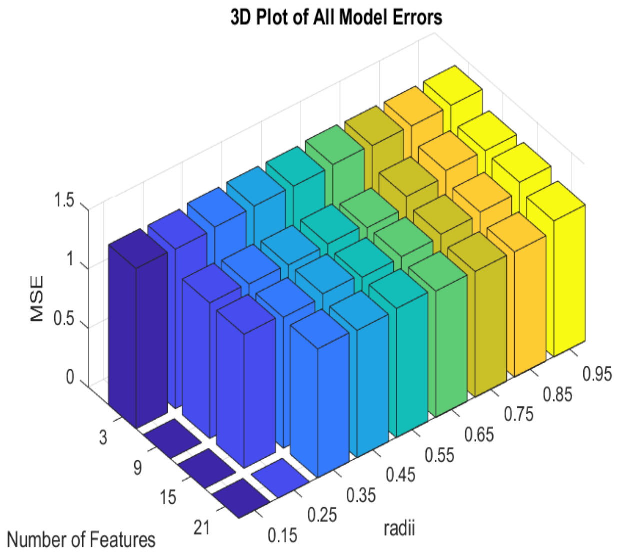


Figure 46: MSE in 3D

The minimum MSE is 1.0624 **for 21 features and radius 0.85**, which we save in `opt_model_dependent.mat`. We observe that no matter the radius and number of features we use, the MSE stays the same. Small improvement exists for higher number of features. Lower radius which means higher number of rules seems to decrease the MSE.

2.5 Optimal Model as Dependent (script `opt_model_dependent.m`)

The optimal model is the one with

- MSE equals 1.2171
- 21 features
- cluster's radius 0.85
- 8 rules

We load the model which we already saved in `opt_model.mat` and train the ANFIS. Results are the following

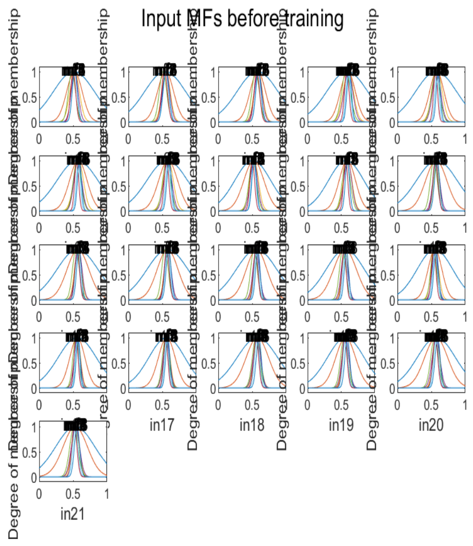


Figure 47: MF before training

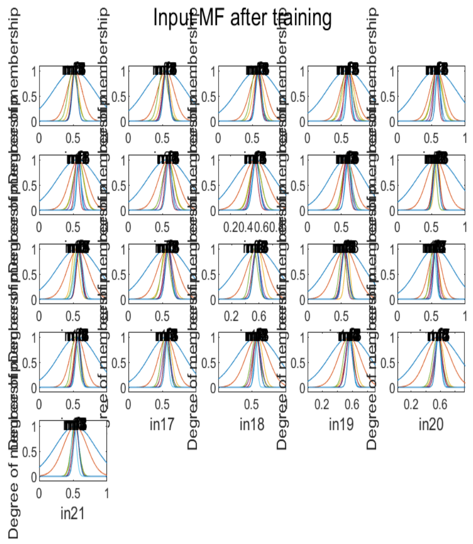


Figure 48: MF after training

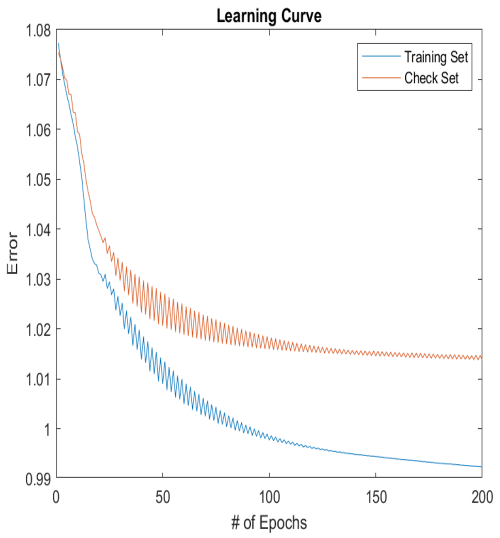
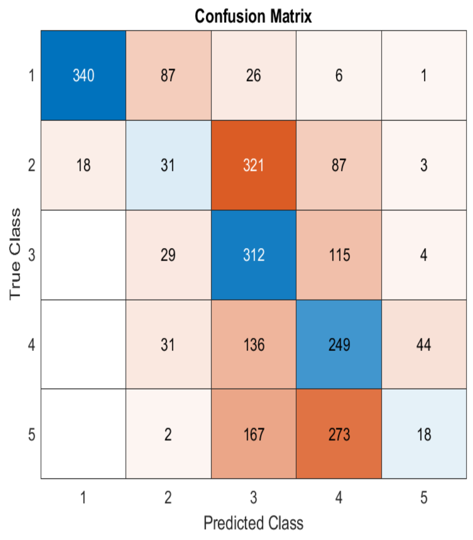


Figure 49: Learning curve (overfitting at epoch ~50)



True Class \ Predicted Class	1	2	3	4	5
1	340	87	26	6	1
2	18	31	321	87	3
3		29	312	115	4
4		31	136	249	44
5		2	167	273	18

Confusion Matrix

Figure 50: Confusion Matrix

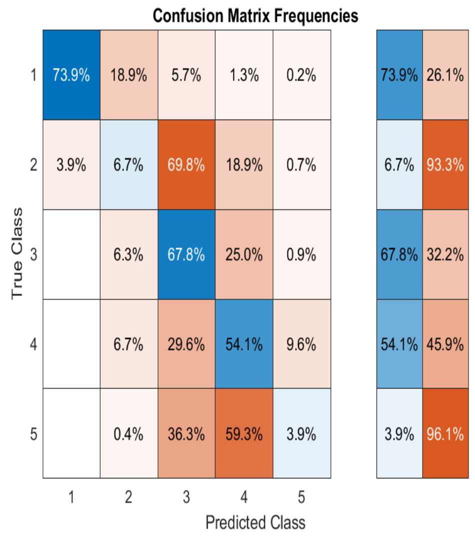


Figure 51: Confusion Matrix - Frequencies

Class Number	1	2	3	4	5	Value
PA	0.9497	0.1722	0.3243	0.3411	*0.2571*	–
UA	0.7391	0.0674	0.6783	0.5413	0.0391	–
OA	–	–	–	–	–	0.4130
$\hat{K}$	–	–	–	–	–	0.2900

Table 19: Metrics

2.6 Conclusion

For the class independent, we observe that the model fails to predict the class with high accuracy. Most of the correct guesses are for class 1 then 4 and 3. The models fails to identify class 5 at all. **Overall accuraccy is 35 %**. Despite the fact that we have 21 features in our model, the number of rules is just 4! For the class dependent, the models appear to be better. The **overall accuracy** is **41 %**. The model predicts class 1 most of the times but it fails to predict class 5 and 2.

List of Figures

List of Tables