Υπολογιστική Νοημοσύνη

Επίλυση προβλήματος παλινδρόμησης με χρήση μοντέλων TSK

Θεόδωρος Κατζάλης
AEM: 9282
`katzalis@ece.auth.gr`

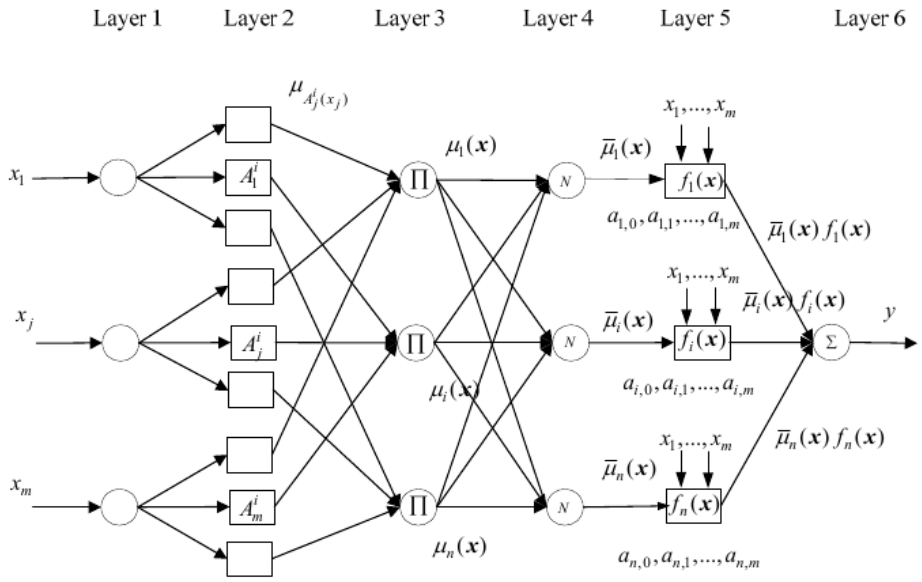
Αριστοτέλειο Πανεπιστήμιο Θεσσαλονικής

February 25, 2023

Περιεχόμενα

1 Εισαγωγή

Η συγκεκριμένη εργασία πραγματεύεται την υλοποίηση ασαφών μοντέλων τύπου Takagi-Sugeno-Kang (TSK) για την επίλυση προβλημάτων παλινδρόμησης σε δυο διαφορετικά datasets με την χρήση `Matlab`. Κάθε τέτοιο πρόβλημα ανάγεται σε ένα πρόβλημα βελτιστοποίησης των παραμέτρων που καθορίζουν το ασαφές μοντέλο. Η αναπαράστασή του με την αρχιτεκτονική ενός νευρωνικού δικτύου (neuro-fuzzy), έχει ως εξής:



Σχήμα 1: Adaptive Neuro Fuzzy Neural Network

Οι παραμέτροι, λοιπόν, που επιθυμούμε να βελτιστοποιήσουμε, αναφέρονται στις μεταβλητές που προσδιορίζουν την μορφή των συναρτήσεων συμμετοχής (Layer 2) αλλά και τις παραμέτρους στο τμήμα συμπερασμού των ασαφών κανόνων (Layer 5). Αναλυτικότερα, ένα ασαφές μοντέλο αρχικά διεγείρεται απο τις εισόδους (Layer 1) x_i για $i = 1..m$ (crisp τιμές). Στη συνέχεια γίνεται η μετάβαση της πληροφορίας των crisp τιμών σε fuzzy σύνολα με την βοήθεια των συναρτήσεων συμμετοχής (Layer 2) και ο ασαφής ελέγχος λαμβάνει υπόσταση με την διαμόρφωση κανόνων (Layer 3). Για τους κανόνες χρησιμοποιήθηκαν οι τεχνικές **grid partitioning** και **subtractive clustering**. Σχετικά με την πρώτη αν για κάθε είσοδο αντιστοιχούν k συναρτήσεις συμμετοχής, για m εισόδους θα έχουμε στο σύνολο $n = k^m$ κανόνες [1]. Ο κανόνας $R^{(i)}, i = 1..n$ για ένα ασαφές μοντέλο TSK με πολυωνυμική συνάρτηση εξόδου είναι:

$R^{(i)}$: IF x_1 is A_1^i AND ... AND x_m is A_m^i THEN $y = f_i(x) = g_{i,0} + g_{i,1}x_1 + ... + g_{i,n}x_m$

Έτσι λοιπόν, το λογικό AND υλοποιείται με την μορφή πολλαπλασιασμού του αποτελέσματος της διέγερσης των συναρτήσεων συμμετοχής (Layer 3), στη συνέχεια γίνεται μια κανονικοποίηση (Layer 4) και στο τέλος, στο τμήμα συμπερασμού, μοντελοποιείται το κομμάτι της συνθήκης THEN.

Η τεχνική που θα ακολουθήσουμε για να εκπαιδεύσουμε το δίκτυο μας και να βρούμε τις βέλτιστες τιμές των παραμέτρων που θα προσδιορίσουν το μοντέλο μας, βασίζεται σε μια υβριδική μέθοδο χρησιμοποιώντας τις τεχνικές back propagation και linear square estimation. Η δεύτερη θα χρησιμοποιηθεί μόνο στο τελευταίο κρυφό στρώμα για την εύρεση των παραμέτρων συμπερασμού ενώ η πρώτη για το τμήμα διαμόρφωσης των συναρτήσεων συμμετοχής. Η συνάρτηση σε `Matlab` που πραγματοποιεί τα παραπάνω είναι η $anfis$.

[1] Αξίζει να σημειωθεί ότι αυτή η τεχνική για μεγάλο αριθμό εισόδων αυξάνει σημαντικά το υπολογιστικό κόστος, το οποίο θα μας απασχολήσει στο δεύτερο πολύ-διάστατο dataset χρησιμοποιώντας subtractive clustering.

2 Εφαρμογή σε απλό dataset

Το πρώτο σύνολο δεδομένων που έγινε μελέτη είναι το `airfoil self noise` (UCL url), το οποίο αποτελείται απο 1503 δείγματα και 6 γνωρίσματα εκ των οποίων 5 εισόδους και 1 έξοδος. Η υλοποίηση αυτής της ανάλυσης μπορεί να βρεθεί στο αρχείο `simluation_simple.m`. Στόχος είναι η πρόβλεψη αυτής της εξόδου, εκπαιδεύοντας το ασαφές μοντέλο με την αρχιτεκτονική ενός νευρωνικού δικτύου.

2.1 Προετοιμασία δεδομένων και διαχωρισμός του dataset

Ο διαχωρισμός των δεδομένων (split) έγινε με την αναλογία 60%-20%-20% (train,validation,test). Τα δεδομένα του validation χρησιμοποιήθηκαν για να αποφύγουμε overfitting. Σχετικά με την κανονικοποίηση, τα δεδομένα μετασχηματίστηκαν στο εύρος 0-1 μέσω unit hypercube. Αξίζει να σημειωθεί οτι το στάδιο της κανονικοποίησης αποδείχτηκε αρκετά σημαντικό. Δοκιμάζοντας εναλλακτικές μεθόδους, όπως για παράδειγμα min-max, τα αποτελέσματα μας είχαν σημαντική απόκλιση. Τελικά, επιλέχθηκε ιδανικότερη η μέθοδος unit hypercude. Ακόμη, για να αποφύγουμε biased σύνολα δεδομένων κατά τον διαχωρισμό, πραγματοποιήσαμε ένα shuffle αυτών πριν τον διαχωρισμό τους.

2.2 TSK μοντέλα

Έγινε μελέτη 4 TSK μοντέλων, των οποίων οι κανόνες διαμορφώθηκαν με grid paritioning (αριθμός κανόνων $=$ αριθμός εισόδων$^{\text{πλήθος συναρτήσεων συμμετοχής}}$) και οι διαφορές τους αφορούν τον αριθμό των συναρτήσεων συμμετοχής για κάθε είσοδο (όμοιος για όλες τις εισόδους) και το τμήμα συμπερασμού. Αναλυτικότερα:

	Πλήθος συναρτήσεων συμμετοχής	Μορφή εξόδου
TSK_model_1	2	Singleton
TSK_model_2	3	Singleton
TSK_model_3	2	Polynomial
TSK_model_4	3	Polynomial

Πίνακας 1: Ταξινόμηση μοντέλων προς εκπαίδευση

Η μορφή των συναρτήσεων συμμετοχής είναι `bell shaped (gbellmf)` με βαθμό επικάλυψης 0.5. Η δημιουργία αυτών των μοντέλων έγινε με την συνάρτηση `genfis`.

2.3 Αξιολόγηση μοντέλων

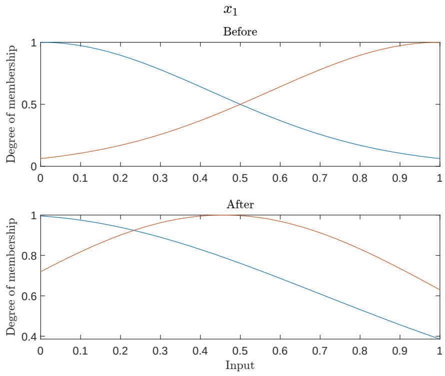
Μετά το πέρας του training (`anfis`), χρησιμοποιώντας όπως είπαμε προηγουμένως την υβριδική μέθοδο (back propagation + least square estimation) έχουμε τα ακόλουθα αποτελέσματα:

	R2	RMSE	NMSE	NDEI
TSK_model_1	0.6831	3.7431	0.3169	0.5629
TSK_model_2	0.82	2.8209	0.18	0.4243
TSK_model_3	**0.9006**	2.0963	0.0994	0.3153
TSK_model_4	0.5790	4.3141	0.4210	0.6488

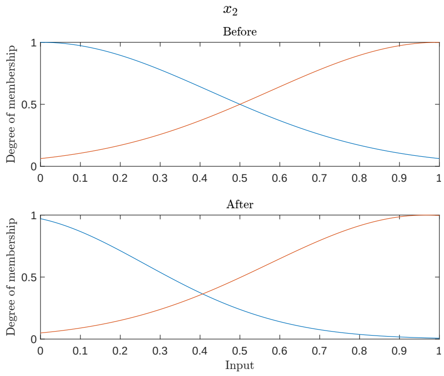
Πίνακας 2: Σύνολο μετρικών αξιολόγησης για τα 4 ασαφή μοντέλα

Στη συνέχεια για κάθε μοντέλο παρουσιάζονται η μεταβολή των συναρτήσεων συμμετοχής για κάθε είσοδο πριν και μετά την εκπαίδευση καθώς και διαγράμματα σφάλματος και απόδοσης των μοντέλων.

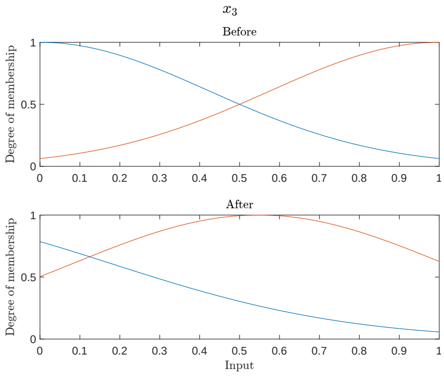
2.3.1 TSK No1



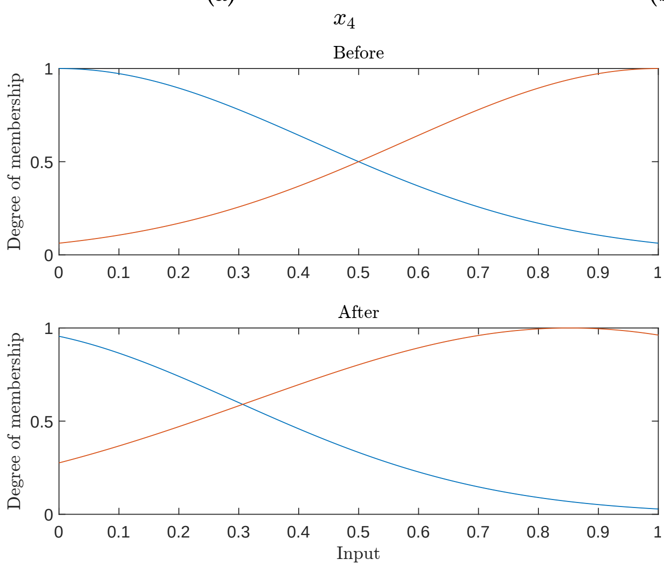
(a)



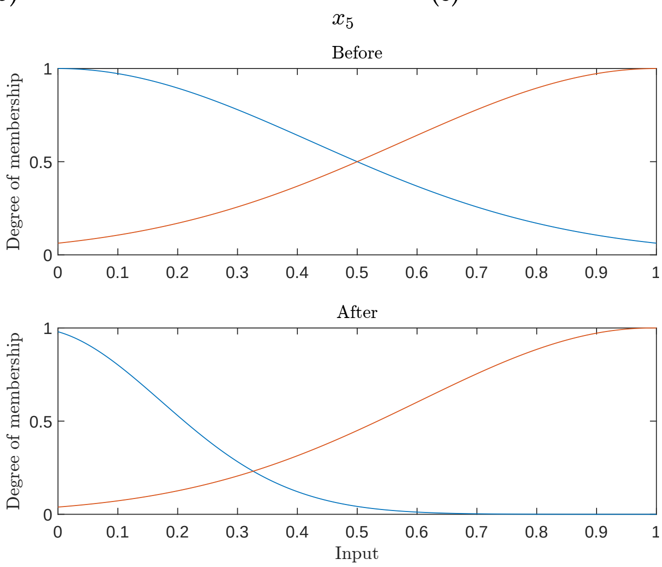
(b)



(c)

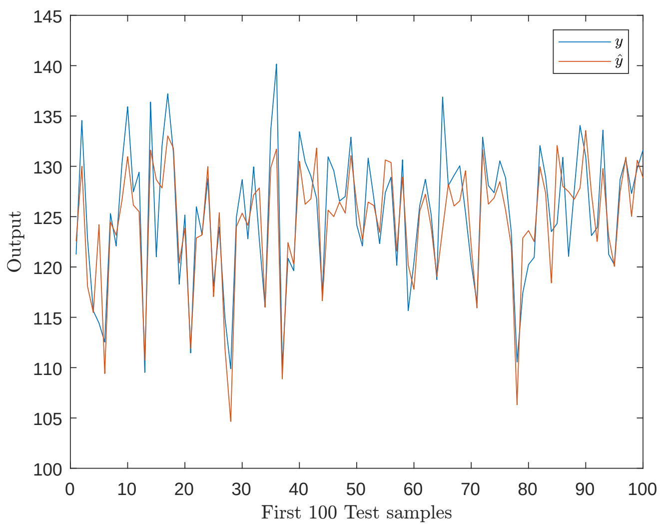


(d)

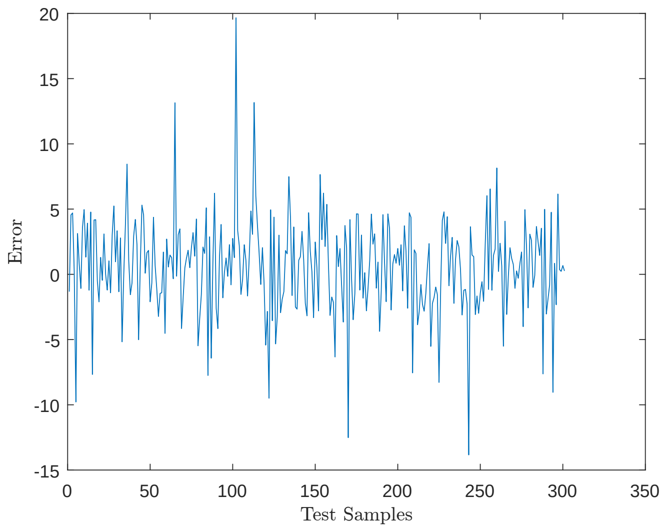


(e)

Σχήμα 2: Συναρτήσεις συμμετοχής πριν και μετά την εκπαίδευση

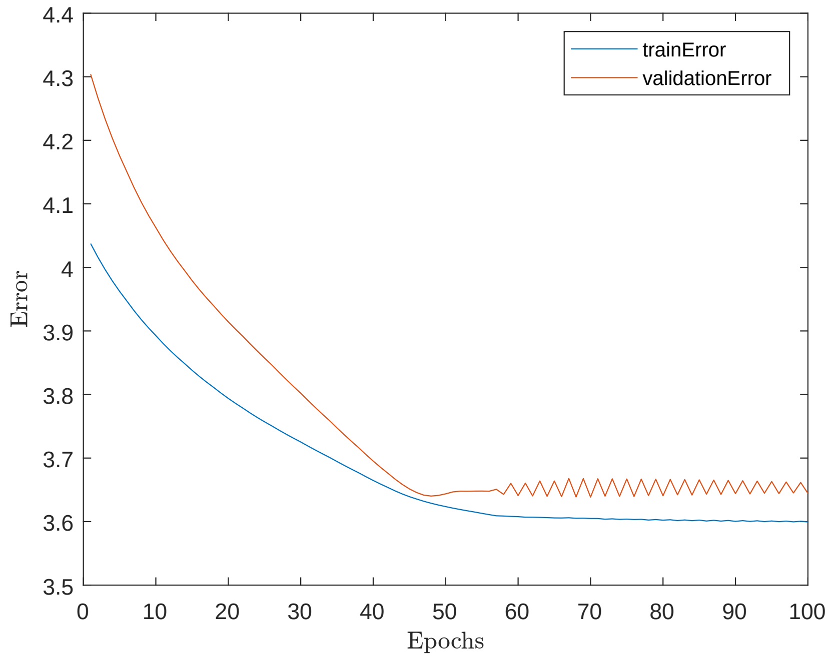


(a)



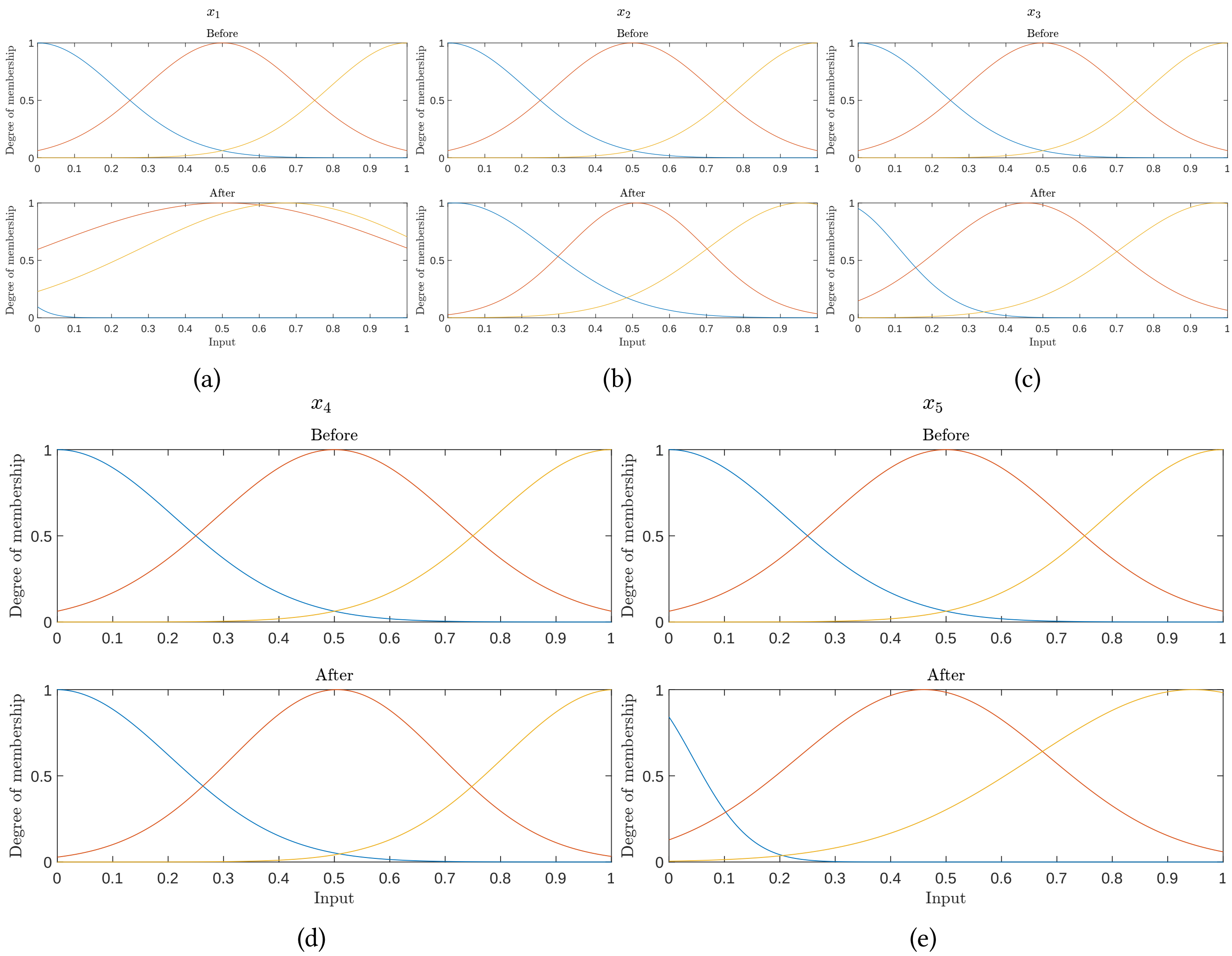
(b) $e = y - \hat{y}$

Σχήμα 3

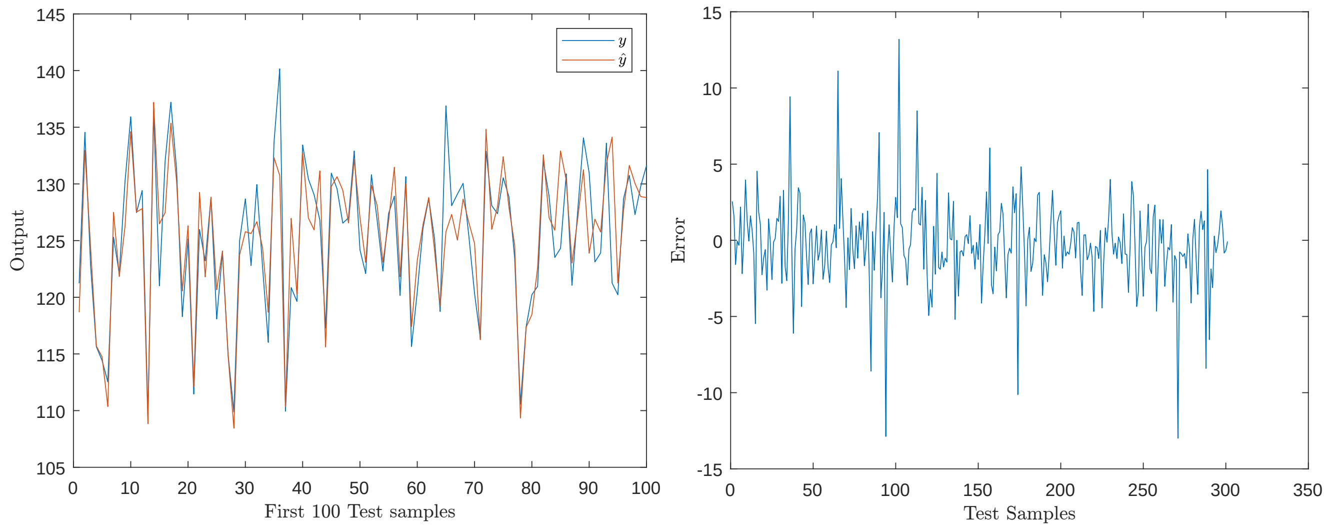


Σχήμα 4: Learning curves

2.3.2 TSK No2

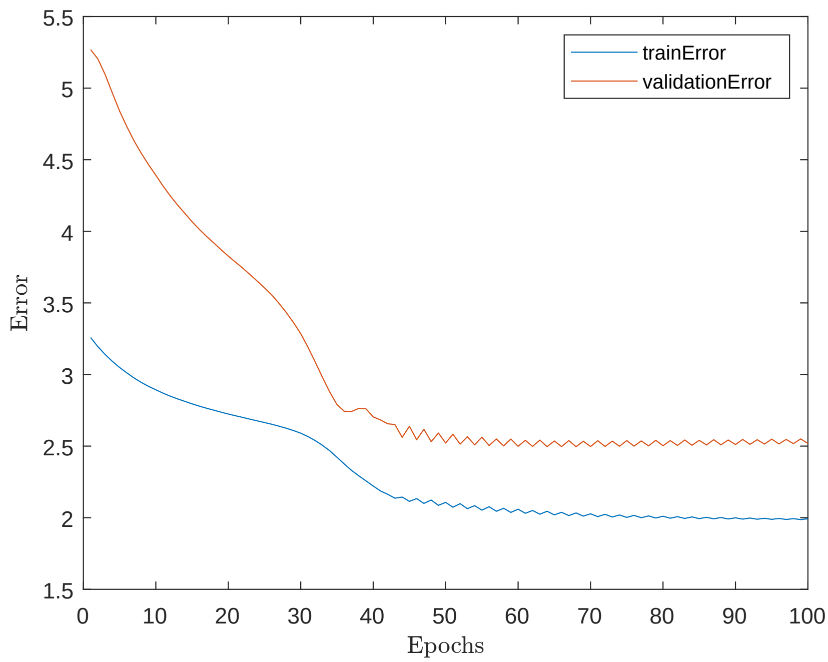


Σχήμα 5: Συναρτήσεις συμμετοχής πριν και μετά την εκπαίδευση



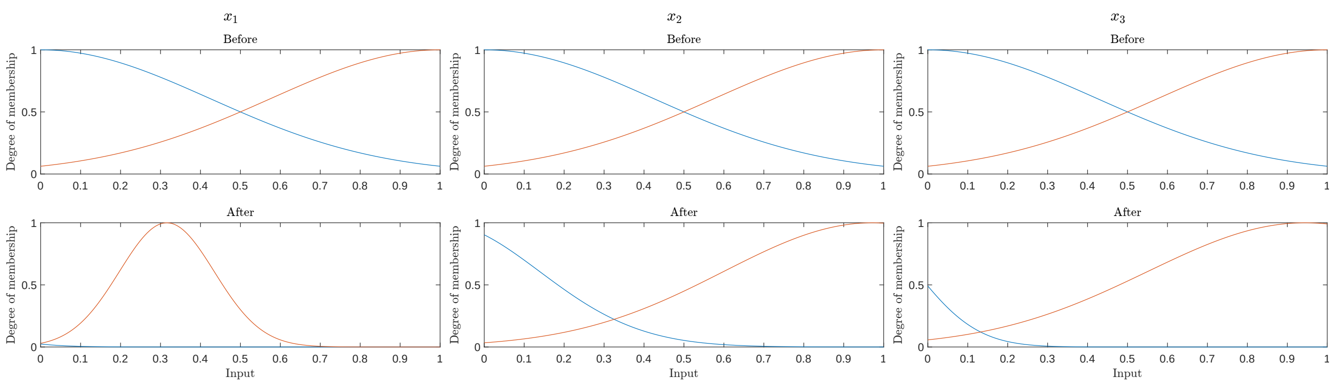
(a) (b) $e = y - \hat{y}$

Σχήμα 6

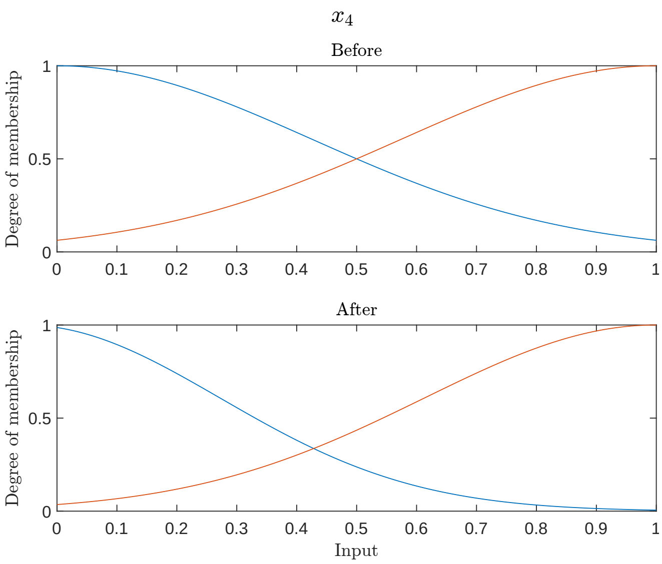


Σχήμα 7: Learning curves

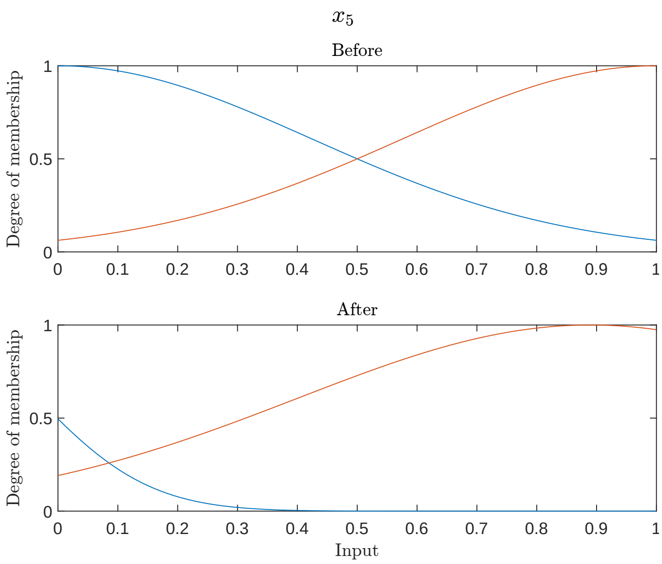
2.3.3 TSK No3



(a) (b) (c)

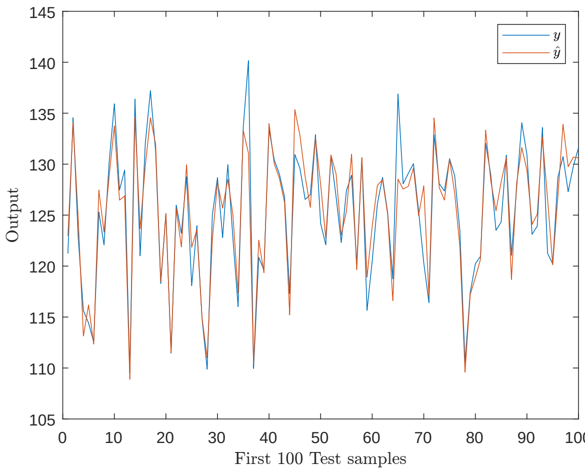


(d)

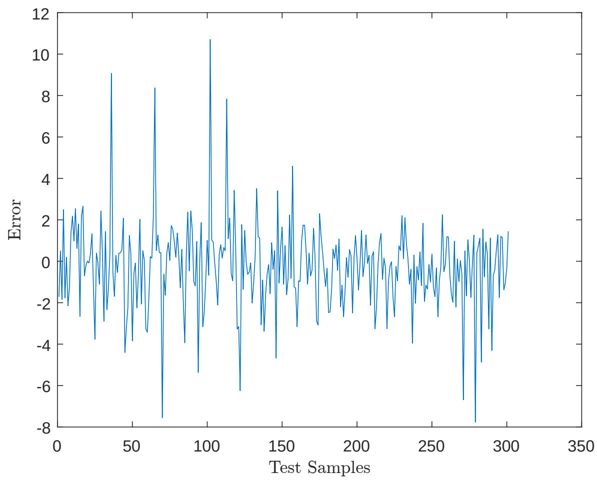


(e)

Σχήμα 8: Συναρτήσεις συμμετοχής πριν και μετά την εκπαίδευση

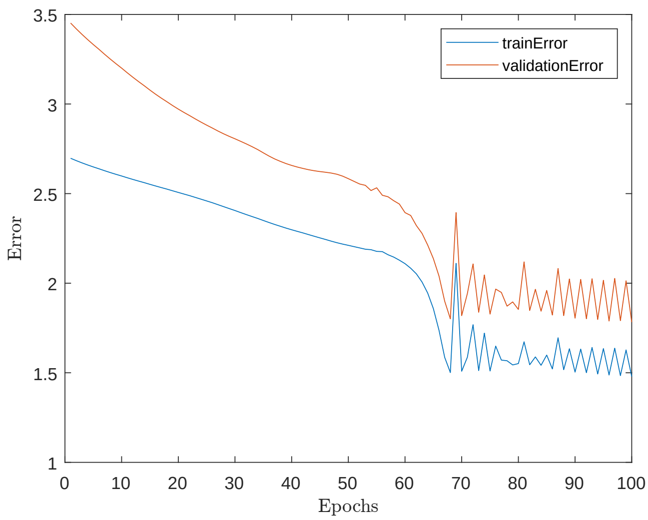


(a)



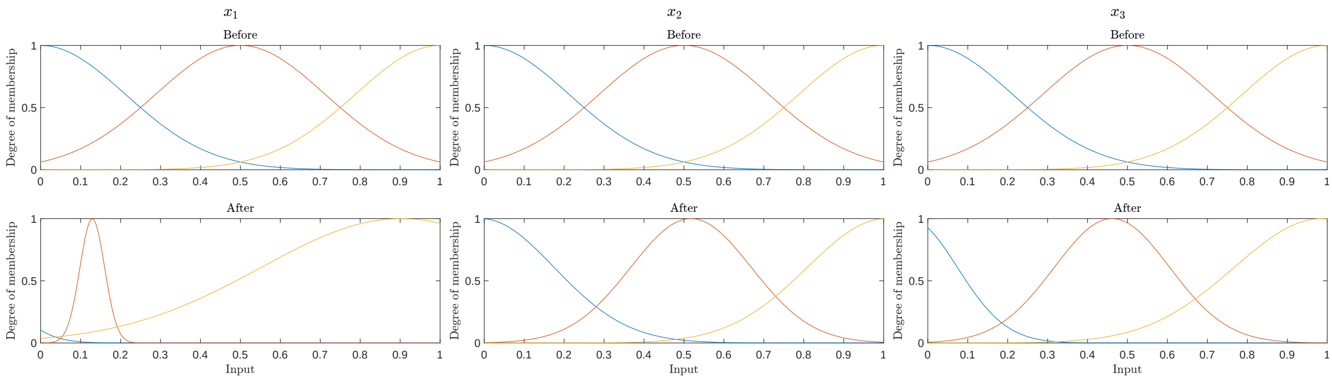
(b) $e = y - \hat{y}$

Σχήμα 9

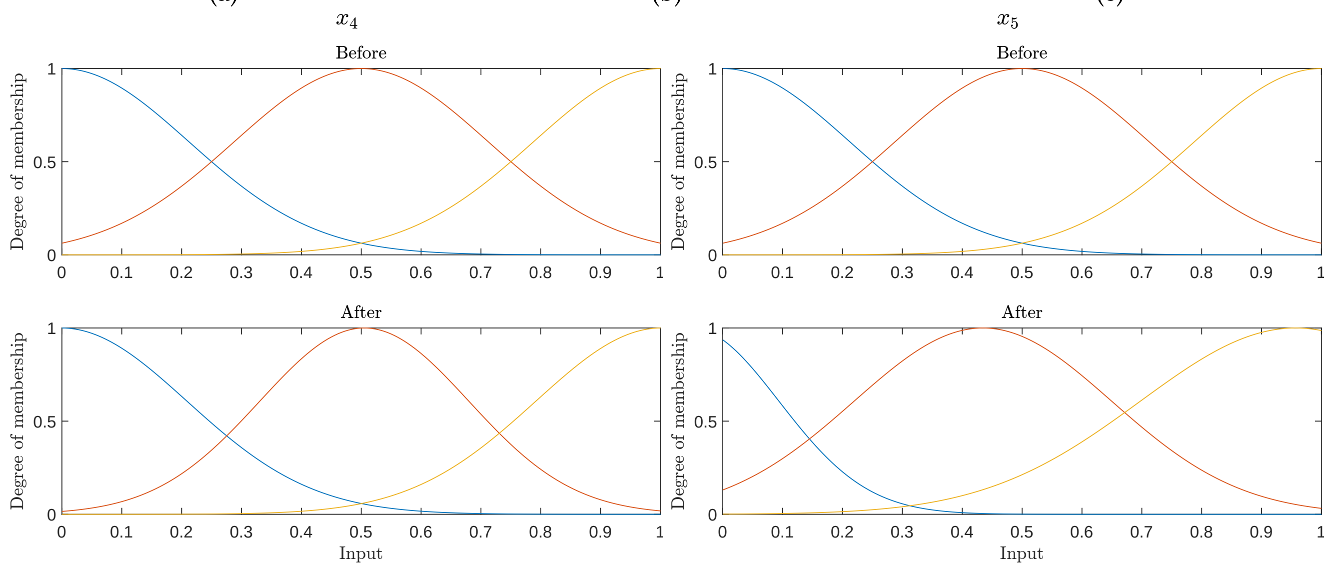


Σχήμα 10: Learning curves

2.3.4 TSK No4

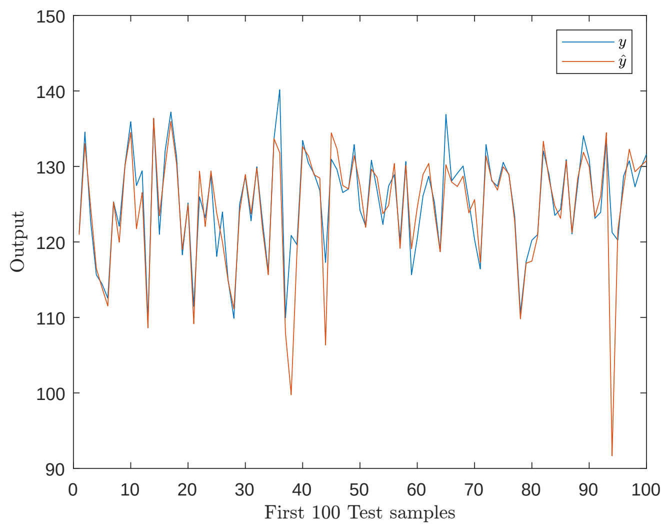


(a) (b) (c)

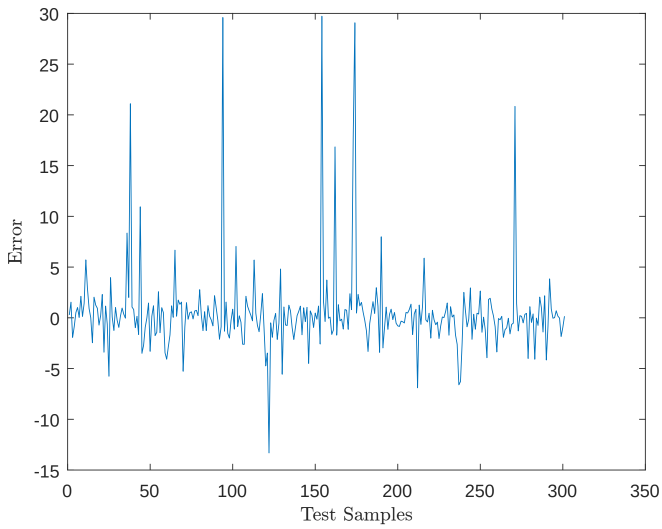


(d) (e)

Σχήμα 11: Συναρτήσεις συμμετοχής πριν και μετά την εκπαίδευση

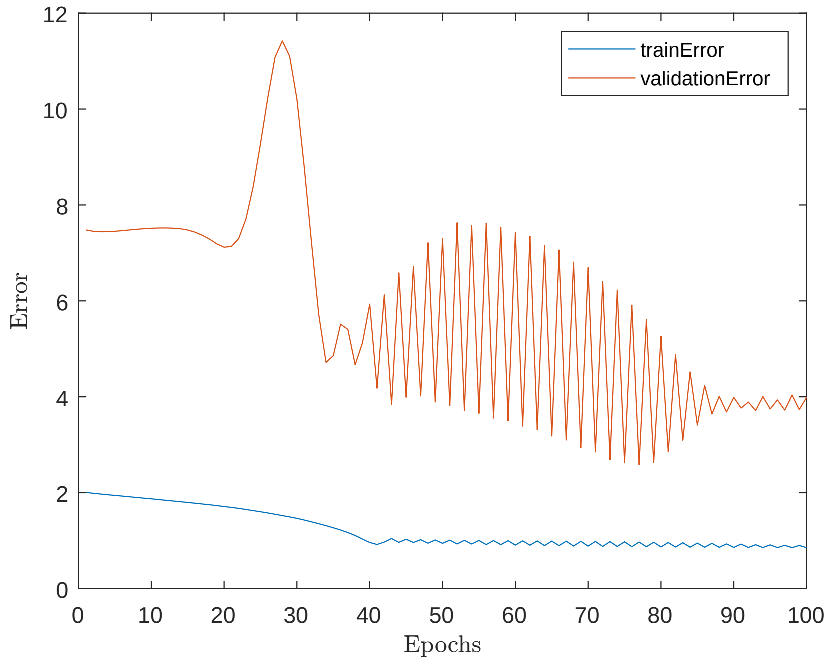


(a)



(b) $e = y - \hat{y}$

Σχήμα 12



Σχήμα 13: Learning curves

Συμπερασματικά, το 3ο μοντέλο παρουσιάζει την καλύτερη απόδοση ($R2 \approx 90\%$) και το 4ο μοντέλο την χειρότερη. Παρατηρώντας τα learning curves, δεν έχουμε κάποιο φαινόμενο overfitting, μιας και δεν παρουσιάζεται σταθερή ανοδική απόκλιση του σφάλματος επιβεβαίωσης σε σχέση με το σφάλμα εκπαίδευσης καθώς αυξάνεται ο αριθμός των εποχών. Σχετικά με το υπολογιστικό κόστος, το 4ο μοντέλο ήταν το πιο απαιτητικό εξαιτίας του μεγάλου αριθμού κανόνων (243) και της πολυωνυμικής συνάρτησης συμπερασμού. Αξίζει ακόμη να σημειωθεί ότι η κακή απόδοση του 4ου μοντέλου θα μπορούσε να αποδοθεί στην αύξηση των νευρώνων στα κρυφά στρώματα αλλά και του μικρού μεγέθους του dataset για να εκπαιδευτεί επιτυχώς. Περισσότεροι νευρώνες είναι ακόμη επιρρεπείς σε overfitting καθώς μπορούν να μάθουν πολύ καλά το σύνολο εκπαίδευσης. Στο 3ο φαίνεται να υπάρχει η χρυσή τομή μεταξύ δεδομένων και νευρώνων.

3 Εφαρμογή σε dataset με υψηλή διαστασιμότητα

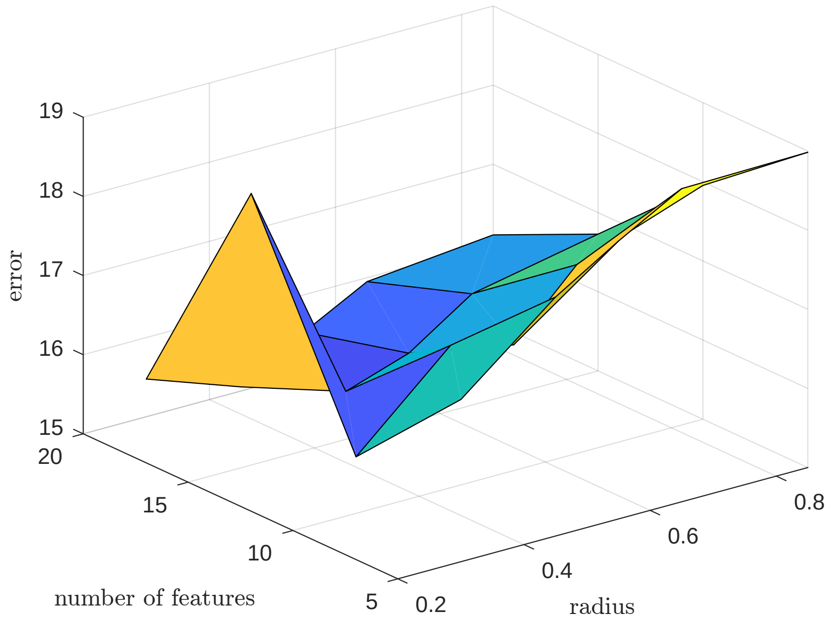
Το δεύτερο σύνολο δεδομένων που έγινε μελέτη είναι το Superconductivity (UCL url) το οποίο αποτελείται απο 21263 δείγματα και 82 γνωρίσματα εκ των οποίων 81 εισόδους και 1 έξοδος (η ανάλυση βρίσκεται στο αρχείο `simulation_multidimensions`). Η ειδοποιός διαφορά αυτού του dataset με το προηγούμενο είναι ο αριθμός των εισόδων, δηλαδή η αύξηση των διαστάσεων του προβλήματος. Αποτέλεσμα αυτού είναι η εκθετική αύξηση του υπολογιστικού κόστους της απλής τεχνικής διαμόρφωσης κανόνων με grid partitioning, όπως έχουμε αναφέρει προηγουμένως. Για να αντιμετωπιστεί πρακτικά λοιπόν αυτή η περίπτωση έγινε εφαρμογή τεχνικών μείωσης α) της διαστασιμότητας (features reduction) αλλά και των β) IF-THEN κανόνων. Για να επιτύχουμε το πρώτο, χρησιμοποιήσαμε την συνάρτηση `relieff` (υλοποιεί τον αλγόριθμο ReliefF) αποδίδοντας ένα σκορ σημαντικότητας σε κάθε feature και για το δεύτερο χρησιμοποιήσαμε την τεχνική subtractive clustering. Η τελευταία δημιουργεί μια προβολή της πληροφορίας των κανόνων σε ένα χώρο σημείων που αντιμετωπίζεται ως unit hypercube στον οποίο χώρο γίνεται προσπάθεια ομαδοποίησης και εντοπισμού των βασικών ομάδων - κέντρων (clusters) των κανόνων.

Αξίζει βέβαια να σημειωθεί ότι οι δύο τεχνικές που αναφέραμε για μείωση της πολυπλοκότητας, εισάγουν δύο νέες παραμέτρους, τον αριθμό των βέλτιστων features (`numFeatures`) και την τιμή της ακτίνας 0-1 (`radius`) ως είσοδο του αλγορίθμου subtractive clustering. Για να βρούμε τον καλύτερο δυνατό συνδυασμό αυτών των δύο (**hyperparameter tuning**) χρησιμοποιήσαμε grid partitioning και για το κομμάτι της αξιολόγησης cross-validation 5-fold. Το εύρος τιμών που μελετήσαμε για αυτές τις δύο παραμέτρους είναι: `numFeatures = [5,10,15,20]` και `radius = [0.3,0.45,0.55,0.65,0.85]`.

Σχετικά με τον διαχωρισμό και την προεπεξεργασία των δεδομένων, είναι όμοια με το προηγούμενο dataset. Αρχικά διαχωρίζουμε τα δεδομένα 60%-20%-%20. Χρησιμοποιούμε το 60% της εκπαίδευσης για το hyperparameter tuning (grid partitioning + cross-validation) με 80%-20% διαχωρισμό και στη συνέχεια βρίσκοντας τις βέλτιστες παραμέτρους (`numFeatures`, `radius`) εκπαιδεύουμε το τελικό μοντέλο μας στο αρχικό split του dataset.

3.1 Hyperparameter tuning

Το μέσο σφάλμα των δοκιμών που έγιναν για το grid που διαμόρφωσε ο αριθμός των features και η ακτίνα του clustering, φαίνεται γραφικά στο ακόλουθο διάγραμμα και αναλυτικότερα στον πίνακα:



Σχήμα 14: Cross validation σφάλματα

numFeatures \ radius	0.3	0.45	0.55	0.65	0.85
5	17.0497	18.0201	18.5122	18.9568	18.9839
10	15.7141	16.7993	16.5891	17.3858	17.9543
15	18.4315	15.6063	15.8729	16.4066	16.7283
20	15.4759	**15.0526**	15.5275	15.9498	16.1079

Πίνακας 3: Μέσο σφάλμα grid partitioning με cross validation k-fold

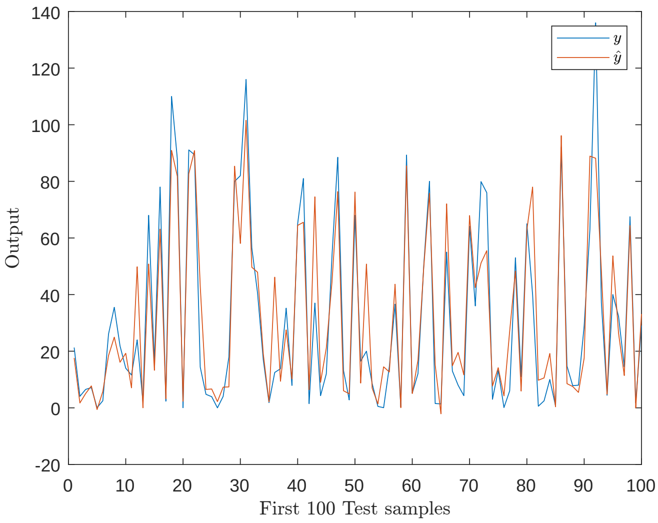
Παρατηρώντας τον πίνακα, φαίνεται να υπάρχει μια αύξηση του μέσου σφάλματος κατα των cross-validation δοκιμών (5-fold) αυξάνοντας την ακτίνα, το οποίο συνεπάγεται και μείωση των κανόνων. Ως προς τον αριθμό των features, δεν υπάρχει μια ξεκάθαρη μονοτονία, ωστόσο για 20 features και 0.45 ακτίνα, έχουμε το καλύτερο δυνατό αποτέλεσμα.

3.2 Αξιολόγηση τελικού μοντέλου

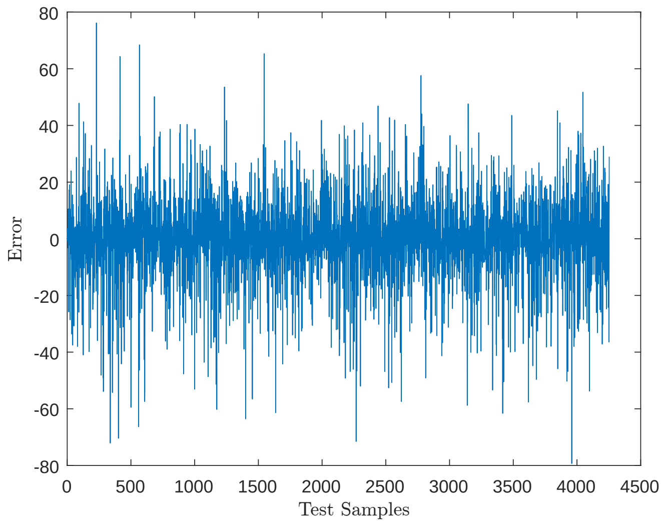
Τελικά, επιλέγοντας τον συνδυασμό με το μικρότερο δυνατό σφάλμα, δηλαδή για το μοντέλο με `numFeatures = 20` και `radius = 0.45` (**8 rules - clusters**) έχουμε:

R2	RMSE	NMSE	NDEI
0.8331	13.8543	0.1669	04085

Πίνακας 4: Μετρικές αξιολόγησης τελικού μοντέλου

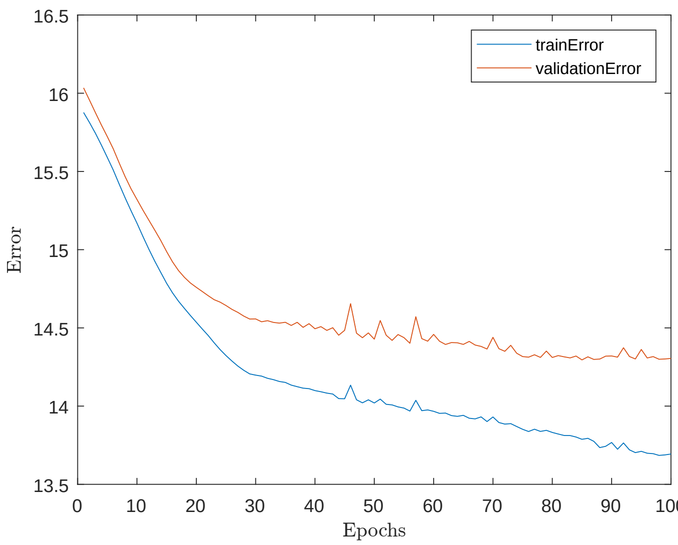


(a) Σύγκριση εξόδου πραγματικής και εκτιμώμενης για τα πρώτα 100 δείγματα αξιολόγησης



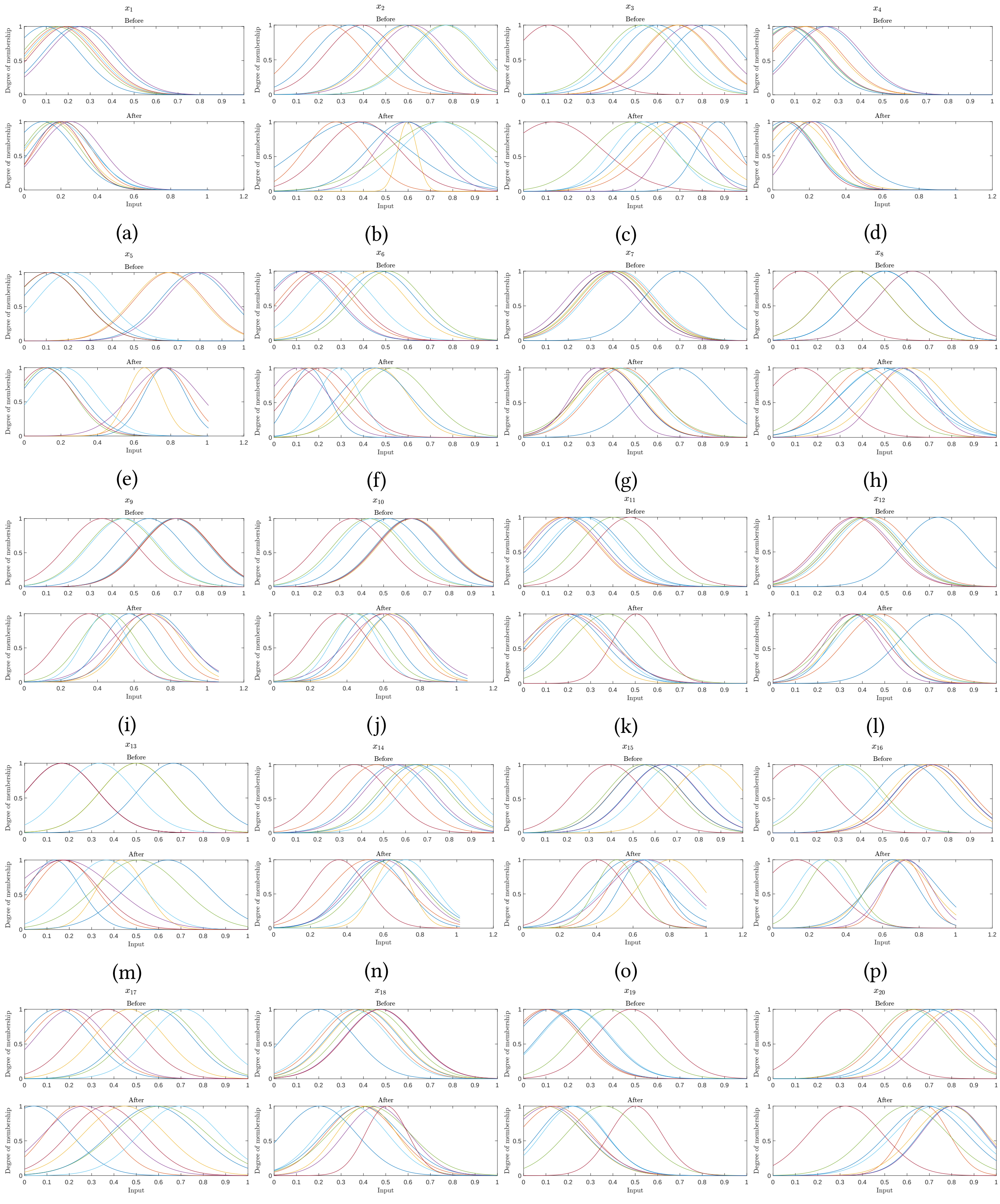
(b) $e = y - \hat{y}$

Σχήμα 15



Σχήμα 16: Learning curves

Αξίζει βέβαια να σημειωθεί ότι για την υψηλή διαστασιμότητα του προβλήματος και την ραγδαία μείωση του υπολογιστικού κόστος με τεχνικές μείωσης αυτής της διαστασιμότητας, έχουμε μια ικανοποιητική απόδοση του τελικού μας μοντέλου ($R2 \approx 83\%$). Είναι ελαφρά χειρότερη απο το 3ο καλύτερο μοντέλο του 1ου dataset, στο οποίο όμως εφαρμόσαμε την πλήρη διαμόρφωση κανόνων με grid partitioning. Σχετικά με τα learning curves δεν παρατηρείται κάποιο φαινόμενο overfitting.



(a) (b) (c) (d) (e) (f) (g) (h) (i) (j) (k) (l) (m) (n) (o) (p) (q) (r) (s) (t)

Σχήμα 17: Συναρτήσεις συμμετοχής πριν και μετά την εκπαίδευση για τις 20 επιλεγμένες εισόδους

4 Matlab

Ο κώδικας για την υλοποίηση των παραπάνω μπορεί να βρεθεί παρακάτω:

4.1 simulation_simple.m

```
clear; close;

% filter I/O
getInput = @(data) data(:, 1:end - 1);
getOutput = @(data) data(:, end);

% prepare data
[trainData, checkData, testData] = prepareData('airfoil_self_noise.dat');

% create fuzzy models
opt = genfisOptions('GridPartition');
opt.InputMembershipFunctionType = 'gaussmf';
opt.OutputMembershipFunctionType = 'constant';
opt.NumMembershipFunctions = 2;
tsk(1) = genfis(getInput(trainData), getOutput(trainData), opt);
opt.NumMembershipFunctions = 3;
tsk(2) = genfis(getInput(trainData), getOutput(trainData), opt);
opt.OutputMembershipFunctionType = 'linear';
opt.NumMembershipFunctions = 2;
tsk(3) = genfis(getInput(trainData), getOutput(trainData), opt);
opt.NumMembershipFunctions = 3;
tsk(4) = genfis(getInput(trainData), getOutput(trainData), opt);

% training
for i = 1:numel(tsk)
    opt = anfisOptions('InitialFIS', tsk(i), 'EpochNumber', 100, ...
        'ValidationData', checkData);
    opt.DisplayANFISInformation = 0;
    opt.DisplayErrorValues = 0;
    opt.DisplayStepSize = 0;
    opt.DisplayFinalResults = 0;
    [trainFis, trainError, ¬, checkFis, checkError] = anfis(trainData, opt);

    % membership functions before vs after training
    for j = 1:numel(checkFis.Inputs)
        figure
        subplot(211)
        % before
        [xmf, ymf] = plotmf(tsk(i), 'input', j);
        plot(xmf, ymf); ylabel('Degree of membership', 'Interpreter', 'Latex')
        title('Before', 'Interpreter', 'Latex')
        subplot(212)
        % after training
        [xmf, ymf] = plotmf(checkFis, 'input', j);
        plot(xmf, ymf); xlabel('Input', 'Interpreter', 'Latex'); ylabel('Degree ...
            of membership', 'Interpreter', 'Latex')
        title('After', 'Interpreter', 'Latex')
        textTitle = ['$x_{' int2str(j) '}$'];
        sgtitle(textTitle, 'Interpreter', 'Latex')
        %exportgraphics(gcf, [int2str(j) '_tsk3.pdf'],'ContentType','Vector')
    end

    %showrule
    %showrule(checkfis)

    %learning curves
    figure
    plot([trainError checkError]); xlabel('Epochs', 'Interpreter', 'Latex'); ...
        ylabel('Error', 'Interpreter', 'Latex'); legend('trainError', ...
        'validationError')
```

```
    %exportgraphics(gcf, 'learning_curves_tsk3.pdf','ContentType','Vector')

    % evaluation, metrics
    yHat = evalfis(checkFis, getInput(testData));
    y = getOutput(testData);
    error = y - yHat;
    r2 = 1 - sum((y - yHat).^2) / sum((y - mean(y)).^2)
    rmse = sqrt(mse(yHat, getOutput(testData)))
    nmse = 1 - r2
    ndei = sqrt(nmse)

    % a sample of 100 elements of the model trying to fit to the data
    figure
    plot([y(1:100) yHat(1:100)]); xlabel('First 100 Test samples', ...
        'Interpreter', 'Latex'); ylabel('Output', 'Interpreter', 'Latex'); ...
        legend('$y$', '$\hat{y}$', 'Interpreter', 'Latex')
    %exportgraphics(gcf, 'prediction_real_tsk3.pdf','ContentType','Vector')

    % prediction errors
    figure
    plot(error); xlabel('Test Samples', 'Interpreter', 'Latex'); ylabel('Error', ...
        'Interpreter', 'Latex')
    %exportgraphics(gcf, 'prediction_real_error_tsk3.pdf','ContentType','Vector')
end
```

4.2 simulation_multidimensional.m

```
clear; close

% filter I/O
getInput = @(data) data(:, 1:end - 1);
getOutput = @(data) data(:, end);

% how to split data and cross validation:
% https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7
% ...
    https://stackoverflow.com/questions/49160206/does-gridsearchcv-perform-cross-validation

% 60-20-20 split (training,validation,testing)
[trainData, checkData, testData] = prepareData('superconductivity.csv');

% grid search
numFeatures = [5 10 15 20];
radius = [0.3 0.45 0.55 0.65 0.85];
meanError = zeros(numel(numFeatures), numel(radius));

% feature selection
numNeighbors = 10;
ranks = relieff(getInput(trainData), getOutput(trainData), numNeighbors);

% k-fold cross validation split
k = 5;
rng(10);
out = getOutput(trainData);
cv = cvpartition(numel(out), 'Kfold', k);

%% Hyperparameter tuning cross validation - grid search
for i = 1:numel(numFeatures)
    for j = 1:numel(radius)
        for kth = 1:k
            trainIdx = find(cv.training(kth) == 1);
```

```
            checkIdx = find(cv.test(kth) == 1);
            trainFilt = [trainData(trainIdx, ranks(1:numFeatures(i))) ...
                out(trainIdx)];
            checkFilt = [trainData(checkIdx, ranks(1:numFeatures(i))) ...
                out(checkIdx)];

            % create fuzzy inference system
            opt = genfisOptions('SubtractiveClustering');
            opt.ClusterInfluenceRange = radius(j);
            fis = genfis(getInput(trainFilt), getOutput(trainFilt), opt);
            fprintf("Fold: %d, NumFeature: %d, Radius: %0.2f, Rules: %d",kth, ...
                numFeatures(i), radius(j),numel(fis.Rules))

            % train model
            opt = anfisOptions('InitialFIS', fis, 'EpochNumber', 100, ...
                'ValidationData', checkFilt);
            opt.DisplayANFISInformation = 0;
            opt.DisplayErrorValues = 0;
            opt.DisplayStepSize = 0;
            opt.DisplayFinalResults = 0;
            [trainFis, trainError, ¬, checkFis, checkError] = anfis(trainFilt, ...
                opt);

            meanError(i, j) = meanError(i, j) + mean(checkError);
            fprintf(", meanError: %0.5f\n",mean(checkError));
        end
        fprintf("Total meanError: %0.5f\n",meanError(i,j));
    end
end

% k-fold mean error
meanError = meanError ./ 5

%% Plots

% stem3, meanerror
figure
surf(radius, numFeatures, meanError); xlabel('radius','Interpreter','Latex'); ...
    ylabel('number of features','Interpreter','Latex')
zlabel('error','Interpreter','Latex')

%% Final model

% pick the optimal hyperparameters
[optimalNumFeaturesIdx, optimalRadiusIdx] = find(meanError == min(meanError(:)));
optimalNumFeatures = numFeatures(optimalNumFeaturesIdx)
optimalRadius = radius(optimalRadiusIdx)

% train for the best hyperparameters (radius, numFeatures)
trainOptimal = [trainData(:, ranks(1:optimalNumFeatures)) getOutput(trainData)];
checkOptimal = [checkData(:, ranks(1:optimalNumFeatures)) getOutput(checkData)];
testOptimal = [testData(:, ranks(1:optimalNumFeatures)) getOutput(testData)];

% create fuzzy inference system
opt = genfisOptions('SubtractiveClustering');
opt.ClusterInfluenceRange = optimalRadius;
fisOptimal = genfis(getInput(trainOptimal), getOutput(trainOptimal), opt);
fprintf("Rules: %d\n",numel(fisOptimal.Rules))

% train model
opt = anfisOptions('InitialFIS', fisOptimal, 'EpochNumber', 100, ...
    'ValidationData', checkOptimal);
[trainFis, trainError, ¬, checkFis, checkError] = anfis(trainOptimal, opt);
```

```

% evaluation, metrics
yHat = evalfis(checkFis, getInput(testOptimal));
y = getOutput(testOptimal);
error = y - yHat;
r2 = 1 - sum((y - yHat).^2) / sum((y - mean(y)).^2)
rmse = sqrt(mse(yHat, getOutput(testOptimal)))
nmse = 1 - r2
ndei = sqrt(nmse)

%% Plots

% learning curves
figure
plot([trainError checkError]); xlabel('Epochs','Interpreter','Latex'); ...
    ylabel('Error','Interpreter','Latex'); legend('trainError','validationError')

%% membership functions before vs after training
for j = 1:numel(fisOptimal.Inputs)
    figure
    subplot(211)
    % before
    [xmf,ymf] = plotmf(fisOptimal, 'input', j);
    plot(xmf,ymf); ylabel('Degree of membership','Interpreter','Latex')
    title('Before','Interpreter','Latex')
    subplot(212)
    % after training
    [xmf,ymf] = plotmf(checkFis, 'input', j);
    plot(xmf,ymf); xlabel('Input', 'Interpreter', 'Latex'); ylabel('Degree of ...
        membership','Interpreter','Latex')
    title('After','Interpreter','Latex')
    textTitle = ['$x_{' int2str(j) '}$'];
    sgtitle(textTitle,'Interpreter','Latex')
end

%% a sample of 100 elements of the model trying to fit to the data
figure
plot([y(1:100) yHat(1:100)]); xlabel('First 100 Test ...
    samples','Interpreter','Latex'); ylabel('Output','Interpreter','Latex'); ...
    legend('$y$','$\hat{y}$','Interpreter','Latex')

% prediction errors
figure
plot(error); xlabel('Test Samples','Interpreter','Latex'); ...
    ylabel('Error','Interpreter','Latex')
```

4.3 prepareData.m

```
function [trainData, checkData, testData] = prepareData(name_dataset)
    data = load(name_dataset);

    % shuffle
    rng(2);
    shuffle = @(v) v(randperm(length(v)), :);
    data = shuffle(data);

    % split
    idxTrain = round(0.6 * length(data));
    idxCheck = round(0.8 * length(data));
    trainData = data(1:idxTrain, :);
    checkData = data(idxTrain + 1:idxCheck, :);
```

```
    testData = data(idxCheck + 1:end, :);

    % normalize min-max input
    % trainData(:, 1:end - 1) = normalize(trainData(:, 1:end - 1), 'range');
    % checkData(:, 1:end - 1) = normalize(checkData(:, 1:end - 1), 'range');
    % testData(:, 1:end - 1) = normalize(testData(:, 1:end - 1), 'range');

    % normalization unit hypercube
    trnX = trainData(:, 1:end - 1);
    chkX = checkData(:, 1:end - 1);
    tstX = testData(:, 1:end - 1);
    xmin = min(trnX, [], 1);
    xmax = max(trnX, [], 1);
    trnX = (trnX - repmat(xmin, [length(trnX) 1])) ./ (repmat(xmax, ...
        [length(trnX) 1]) - repmat(xmin, [length(trnX) 1]));
    chkX = (chkX - repmat(xmin, [length(chkX) 1])) ./ (repmat(xmax, ...
        [length(chkX) 1]) - repmat(xmin, [length(chkX) 1]));
    tstX = (tstX - repmat(xmin, [length(tstX) 1])) ./ (repmat(xmax, ...
        [length(tstX) 1]) - repmat(xmin, [length(tstX) 1]));
    trainData(:, 1:end - 1) = trnX;
    checkData(:, 1:end - 1) = chkX;
    testData(:, 1:end - 1) = tstX;
end
```