

Μέρος Α

A1)

Βήματα:

- Κάνουμε χρήση του dataset: census 1994. Είναι κατάλληλο για πρόβλημα κατηγοριοποίησης. Λαμβάνει υπόψιν του τα dataset δεδομένα από την απογραφή του έτους 1994 στις Η.Π.Α. Ειδικότερα παρατηρούμε τις εξής στήλες: **age,workclass,fnlwgt,education,education_num,marital_status,occupation,relationship,race,sex,capital_gain,capital_loss,hours_per_week,native_country,salary.**

Με βάση τους παραπάνω παράγοντες, γίνεται ανάλυση του μισθού ενός εργαζομένου και πρόβλεψη του εάν ανέρχεται πάνω ή κάτω από 50K\$.

Αρχείο για load των δεδομένων: A_1_ClassificationLearner.m

```
% Φόρτωση του dataset census1994
load census1994

% Αποθήκευση των δεδομένων σε μια μεταβλητή
data = adultdata;

% Εμφάνιση των πρώτων 20 δειγμάτων
disp(data(1:20, :));

load('trainedModelCL1.mat', 'trainedModelCL1');
FineTree = trainedModelCL1.ClassificationTree;

new_data = data(:, 1:end-1);

predictions1 = predict(FineTree, new_data);

%pie(predictions1)

% -----
load('trainedModelCL2.mat', 'trainedModelCL2');
LogisticRegression = trainedModelCL2;

predictions2 = LogisticRegression.predictFcn(new_data);

%pie(predictions2)

% -----
load('trainedModelCL3.mat', 'trainedModelCL3');
SVM = trainedModelCL3;

predictions3 = SVM.predictFcn(new_data);

%pie(predictions3)

% -----
load('trainedModelCL4.mat', 'trainedModelCL4');
NaiveBayes = trainedModelCL4;
```

```
predictions4 = NaiveBayes.predictFcn(new_data);  
%pie(predictions4)
```

2. Σε αυτό το σημείο θα επιλέξουμε 4 μοντέλα(επιλογή Classifier Type) για εκπαίδευση και πρόβλεψη.
 - 1) Fine Tree
 - 2) Logistic Regression
 - 3) SVM
 - 4) Naïve Bayes

3. Train Selected για το κάθε μοντέλο ξεχωριστά

Χρόνος εκπαίδευσης(σε sec): 9.4,48.3,152.5,6.9

Accuracy-Validation(σε ποσοστό %):83.1,83.0,81.9,82.9

Test All Trained Models:

Accuracy-Test(σε ποσοστό %):85.3,82.8,81.8,83.1

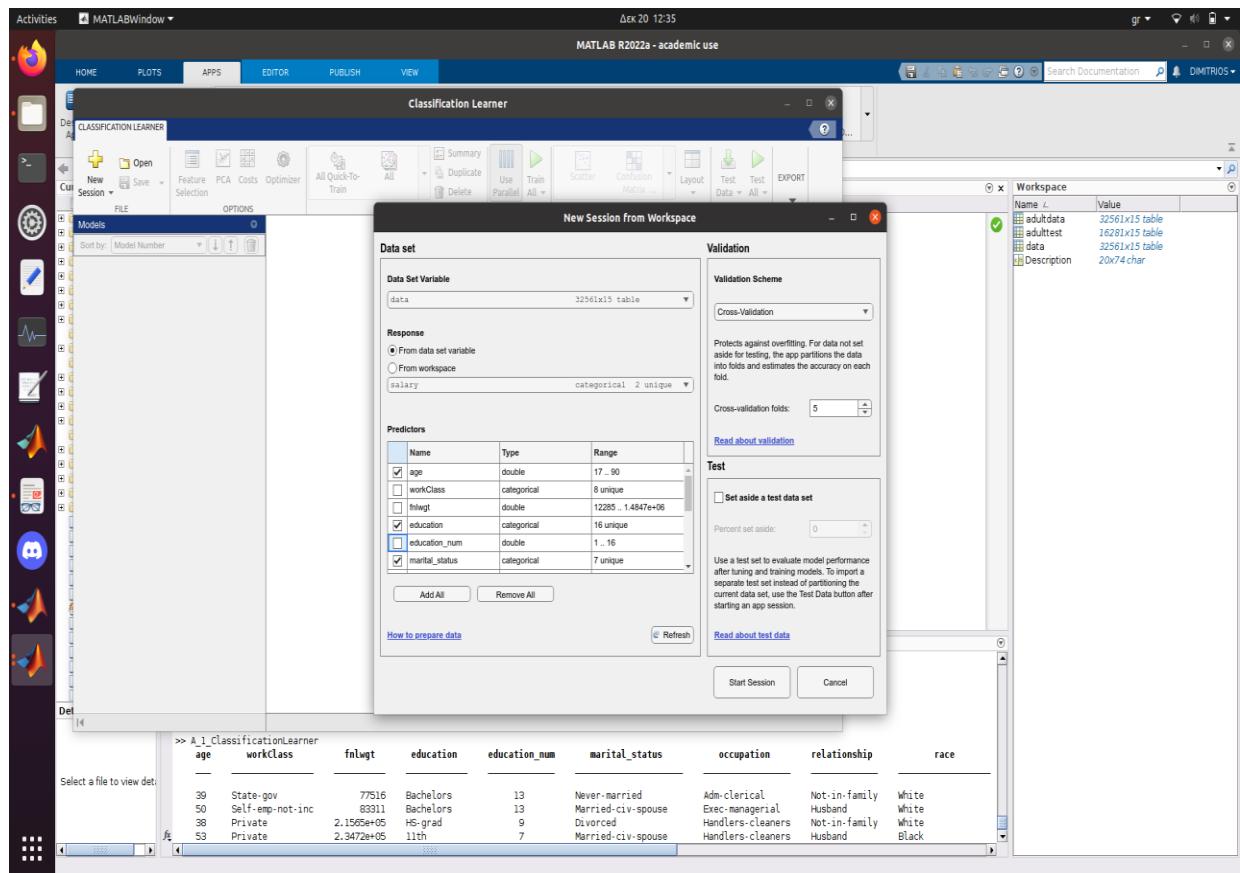
4. Export των μοντέλων στο Workspace(περιέχονται και τα δεδομένα που χρησιμοποιήθηκαν για εκπαίδευση, ώστε να έρθουμε σε επαφή με περισσότερες λεπτομέρειες για κατασκευή του μοντέλου. Επίσης, με αυτή την επιλογή, σε αντίθεση με το Export Compact Model, μας δίνεται η δυνατότητα χρήσης του μοντέλου για περαιτέρω εκπαίδευση και ανάλυση).

Η ίδια διαδικασία επαναλαμβάνεται 4 φορές, δηλαδή μία φορά για κάθε μοντέλο.

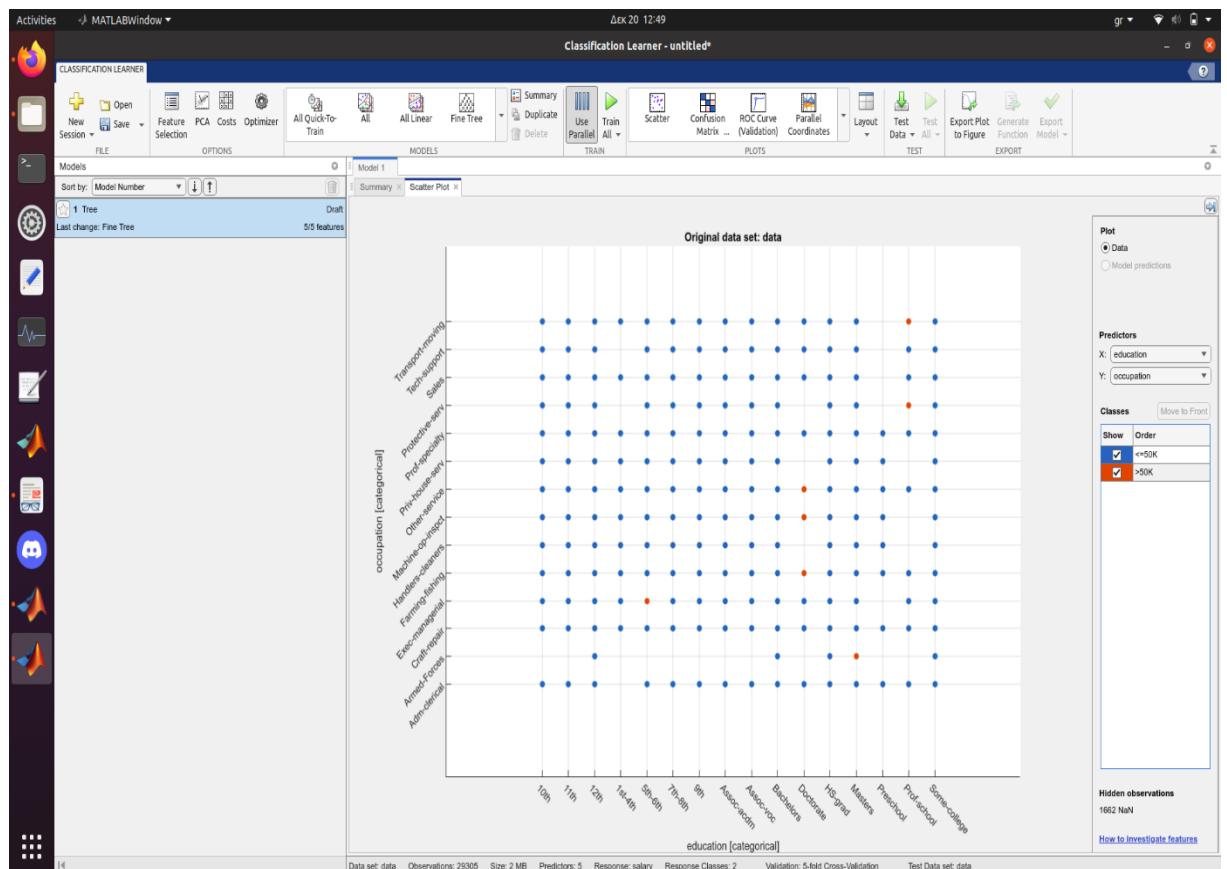
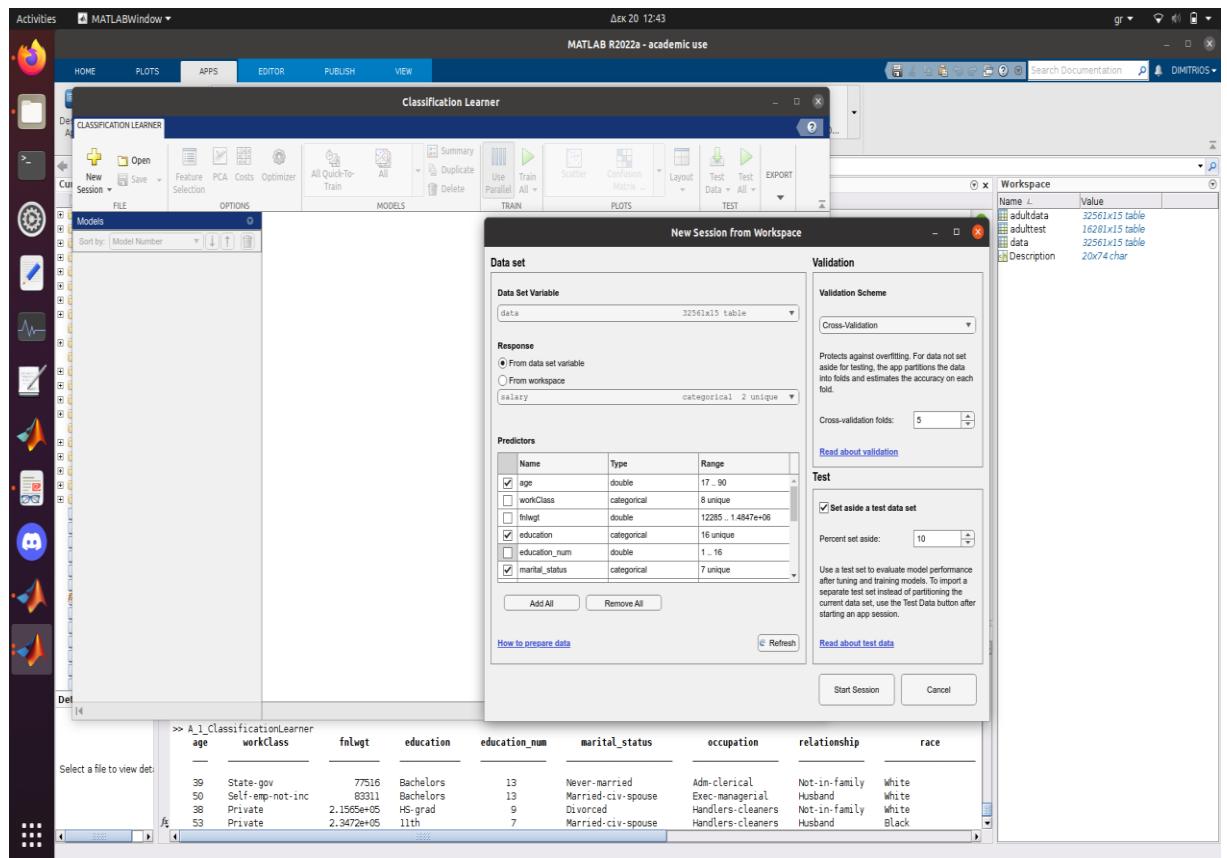
Τέλος, γίνονται προβλέψεις με χρήση της συνάρτησης predictFcn, και με τη βοήθεια των new_data.

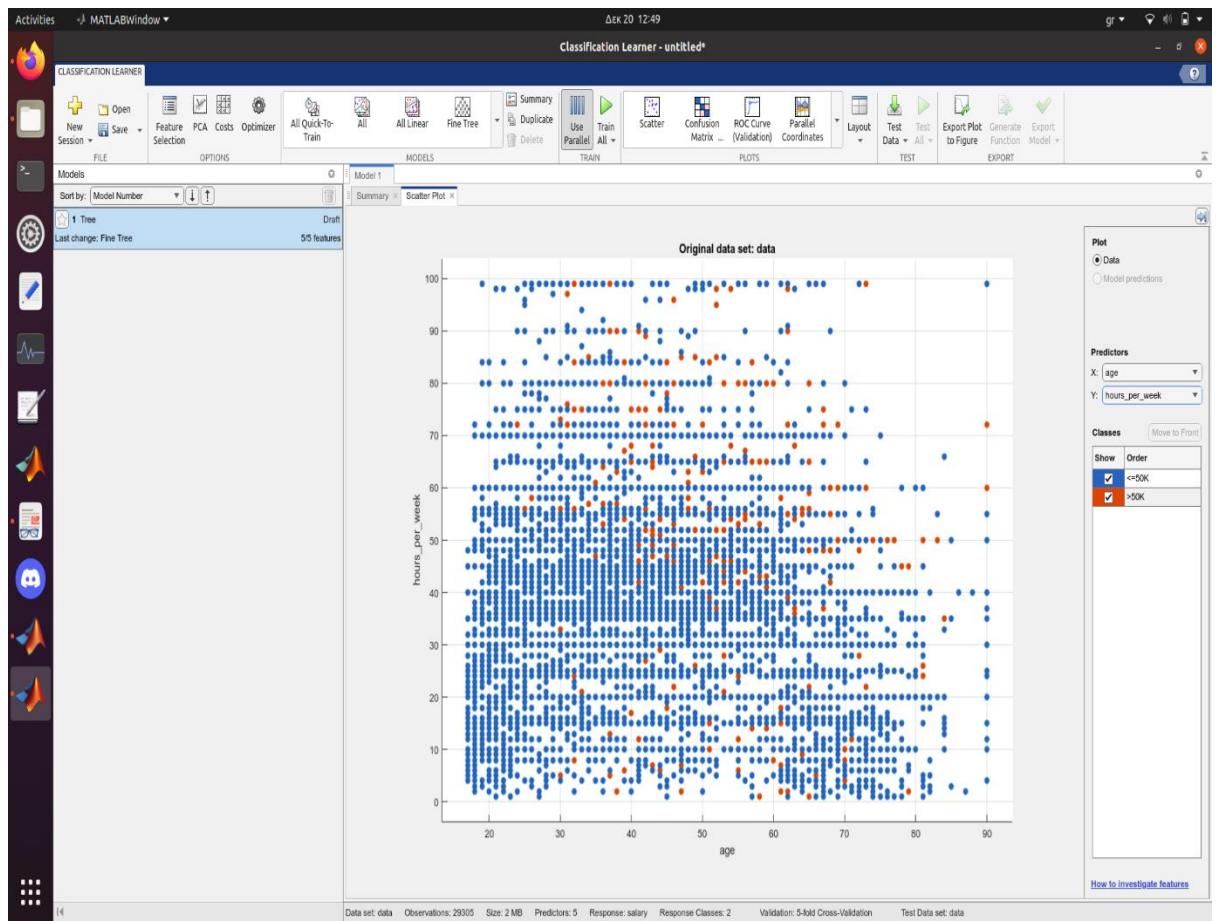
Ανάλυση Αποτελεσμάτων-Γραφημάτων

Οι φωτογραφίες που ακολουθούν αποτελούν screenshots από το περιβάλλον του Matlab, και ειδικότερα από το App:Classification Learner.

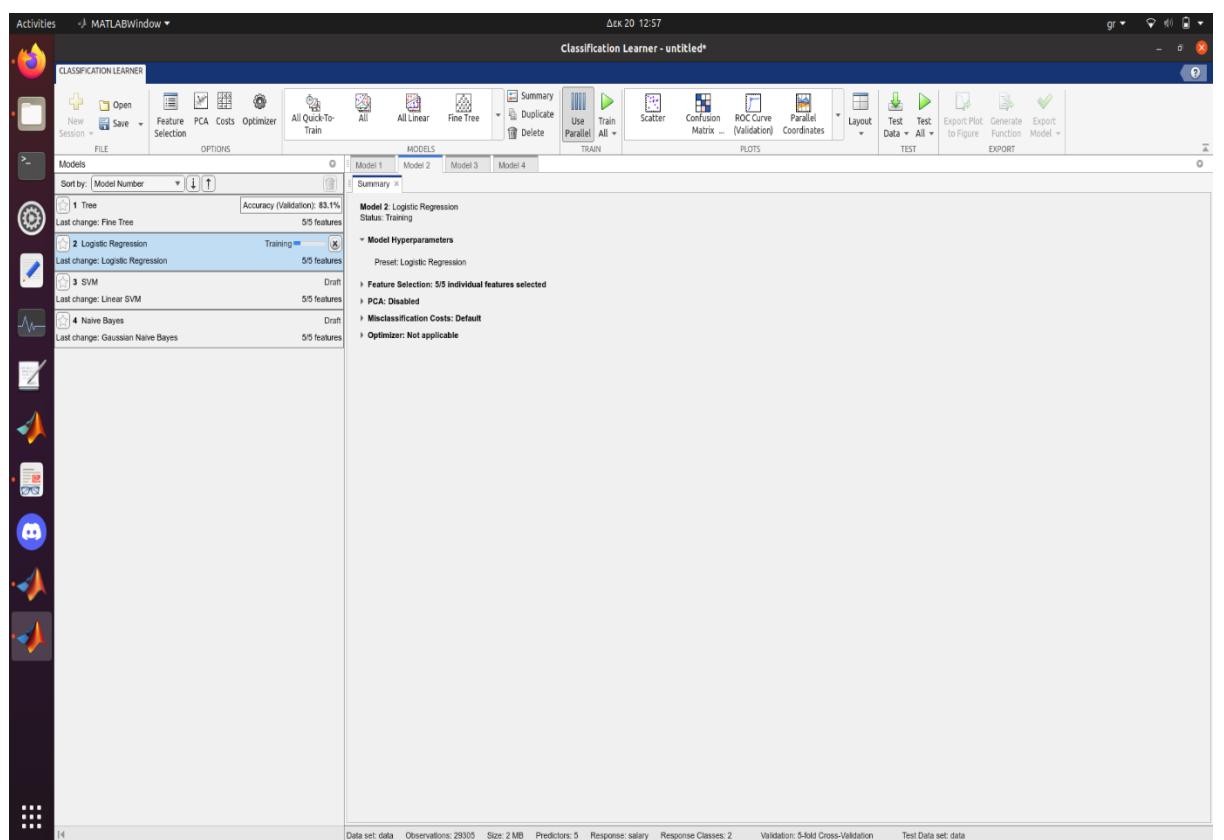
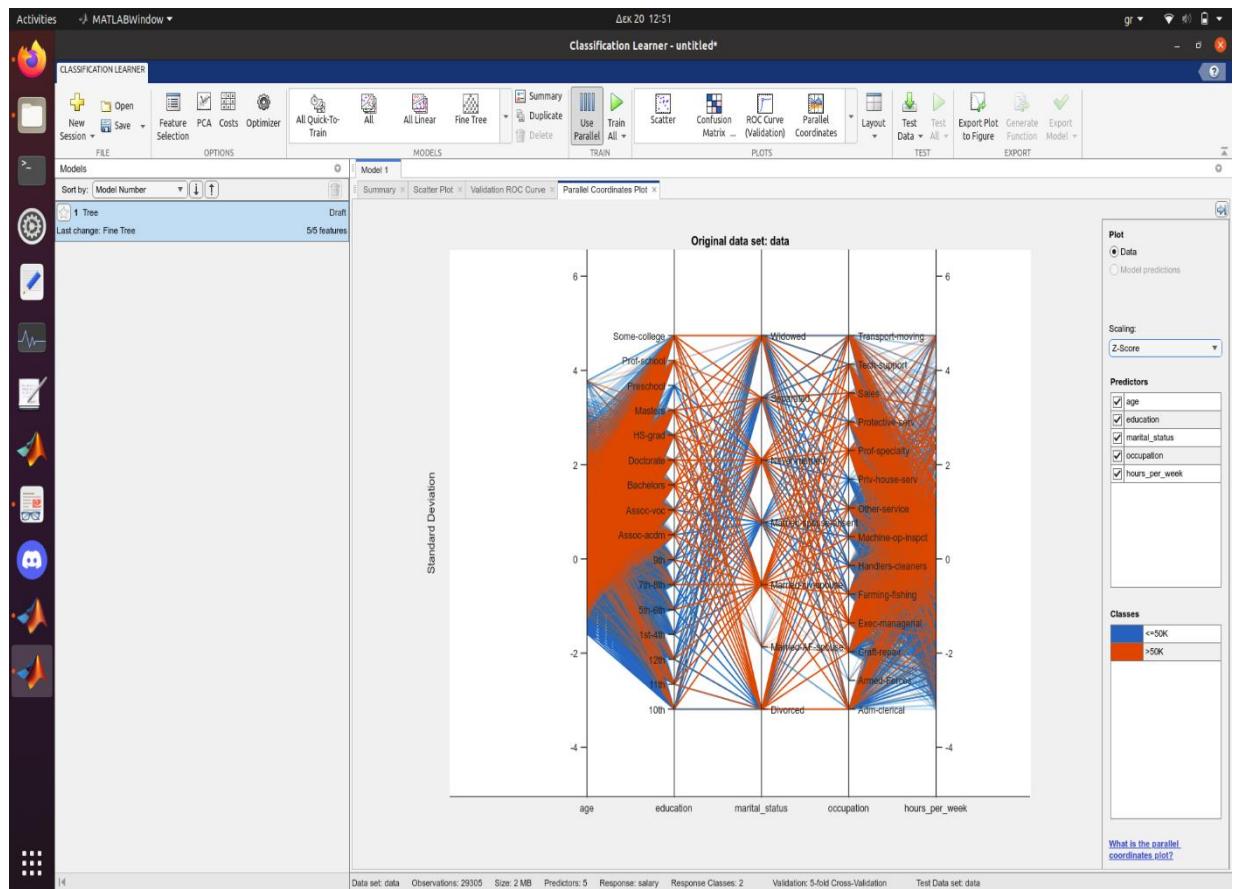


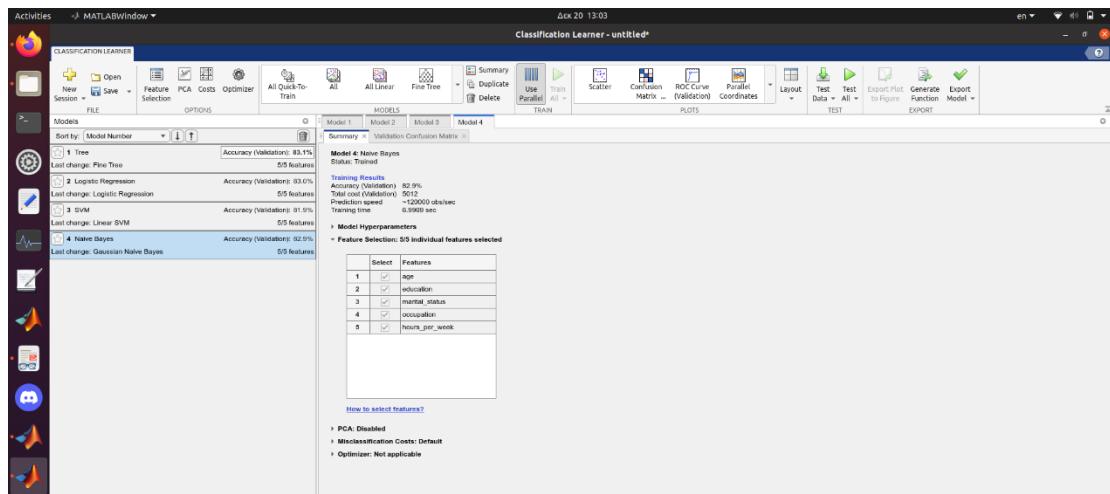
Αρχικά, γίνεται το load των δεδομένων. Ταυτόχρονα, επιλέγονται οι predictors(age, education, marital_status, occupation, hours_per_week). Δεν επιλέχθηκαν όλες τις στήλες του dataset, καθώς κάποιες είναι κοινωνικού χαρακτήρα και τείνουν σε ζητήματα εκτός των ορίων της Μηχανικής Μάθησης. Παράλληλα, επιλέχθηκε k-cross validation, όπου k=5. Συνοπτικά, το δείγμα δεδομένων χωρίζεται σε 5 ομάδες. Έτσι, δημιουργείται μια διαδικασία επαναδειγματοληψίας. Επίσης, δηλώνουμε ότι θέλουμε το 10% του συνόλου των δεδομένων να κρατηθεί για λόγους testing. Τέλος, επιλέγουμε Start Session.





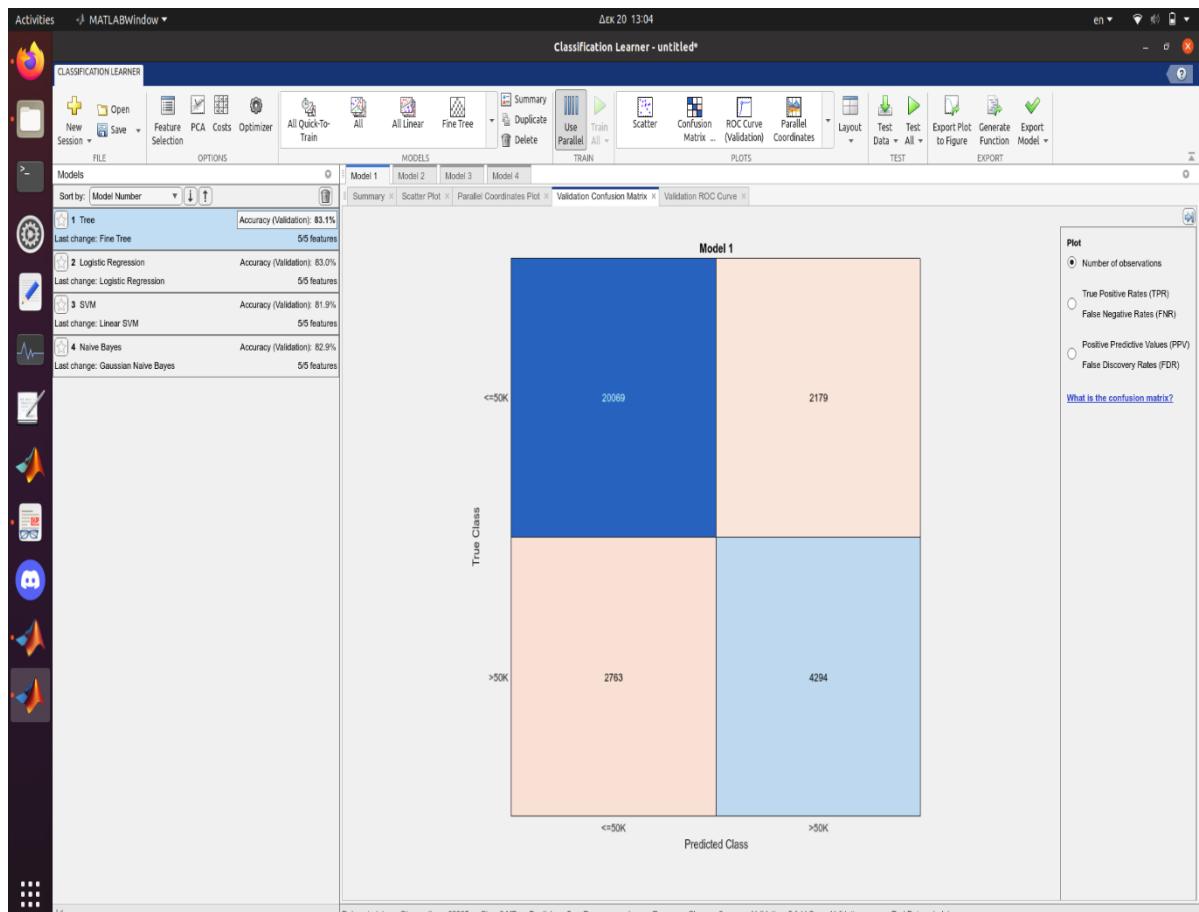
Ακολουθούν 3 διαγράμματα, όπου χρησιμοποιείται το μοντέλο εκμάθησης Fine Tree. Το πρώτο αποτελεί διάγραμμα διασποράς(scatter plot). Στον άξονα x:education και στον άξονα y:occupation. Ενώ, με μπλε κύκλο <<σημαδεύονται>> τα δεδομένα για τα οποία εμφανίζεται salary<=50K\$, αντίθετα με κόκκινο κύκλο τα δεδομένα για τα οποία εμφανίζεται salary>=50K\$. Όπως είναι λογικό τα μπλε κυκλάκια υπερτερούν αριθμητικά(και με μεγάλη διαφορά) από τα κόκκινα. Επιπροσθέτως, προβάλλεται διάγραμμα διασποράς, όπου x άξονας:age και y άξονας:hours_per_week. Παρατηρούμε ότι σε ηλικίες<=30 ετών είναι ελάχιστοι οι εργαζόμενοι που αμείβονται με πάνω από 50K. Ακολουθεί διάγραμμα parallel coordinates. Ως scaling επιλέχθηκε το Z-score, και λαμβάνονται υπόψιν και οι 5 predictors. Ακολουθούν 2 φωτογραφίες, κατά την εκπαίδευση των μοντέλων και μετά το τέλος του.

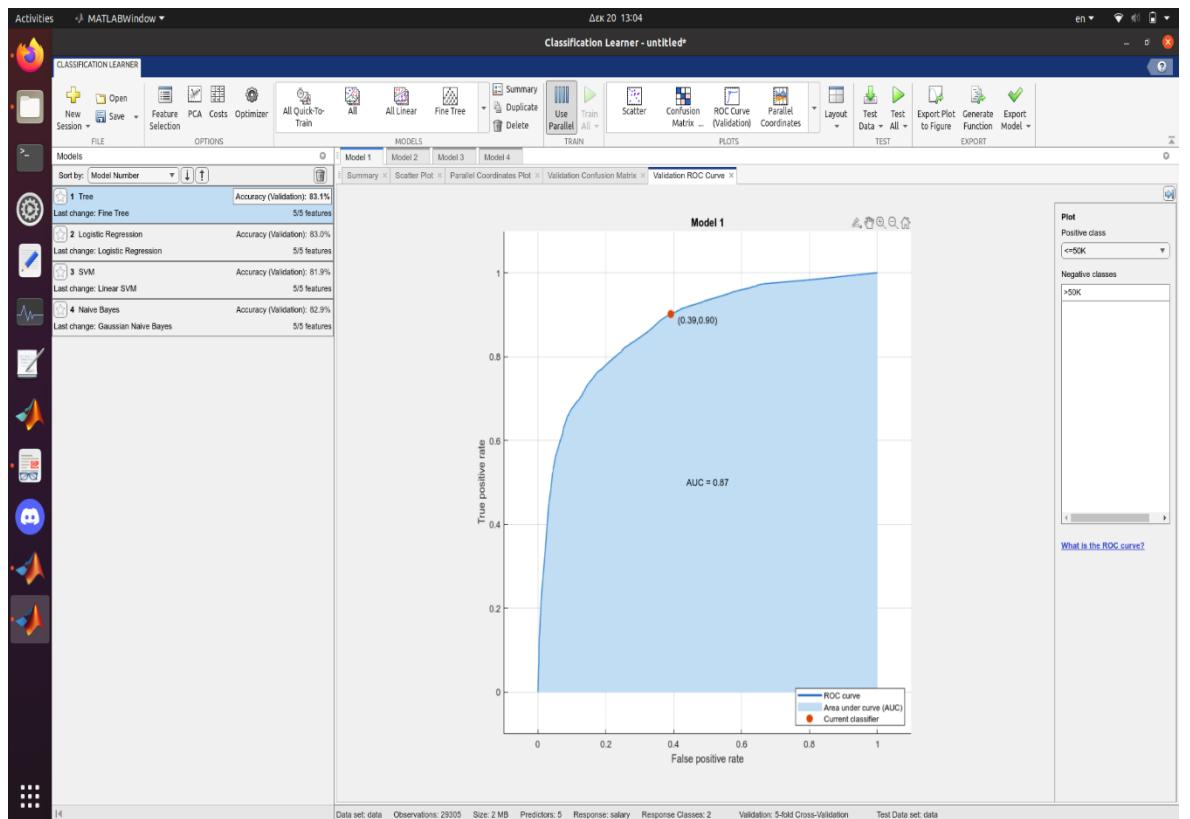




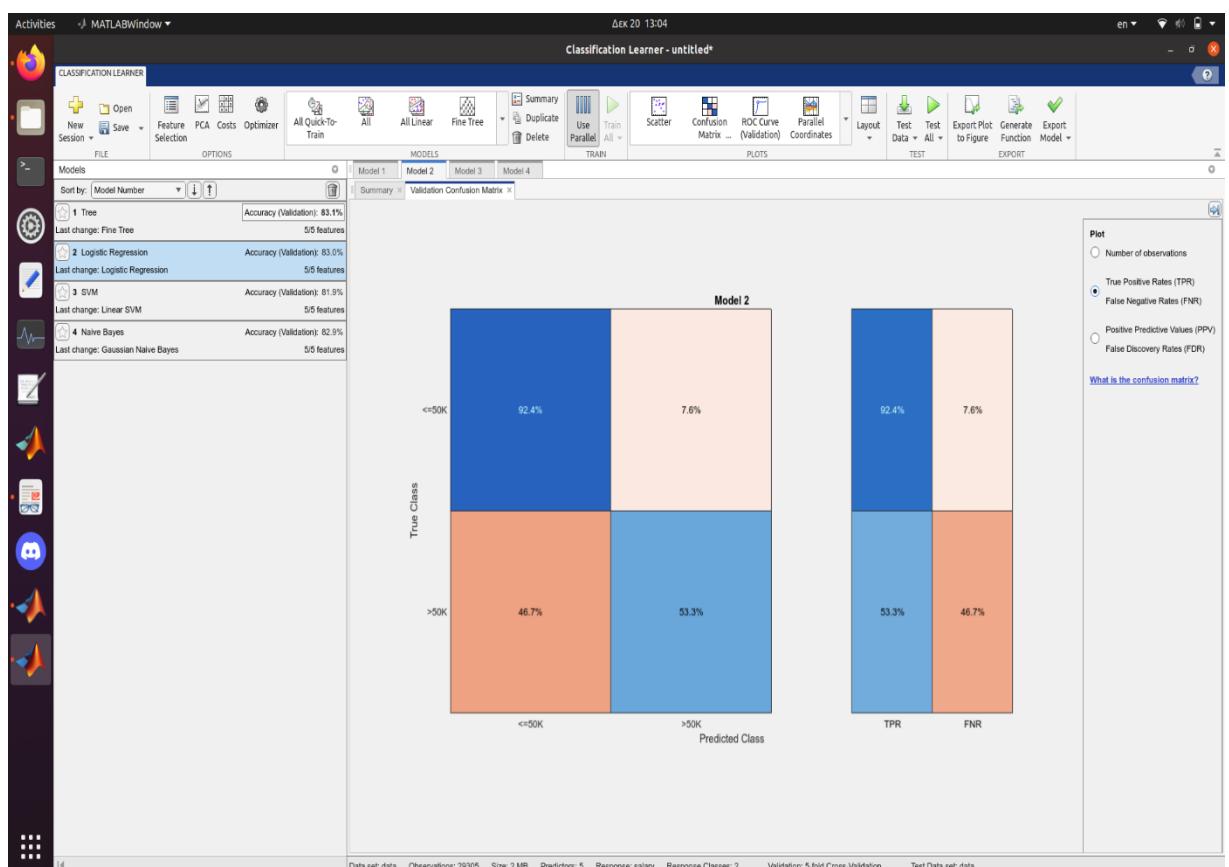
Παρατηρούμε ότι το ταχύτερα εκπαιδευόμενο μοντέλο, αλλά και πιο αποτελεσματικό είναι το Fine Tree. Αξίζει να σημειωθεί πως όλα τα μοντέλα έχουν Accuracy>80%.

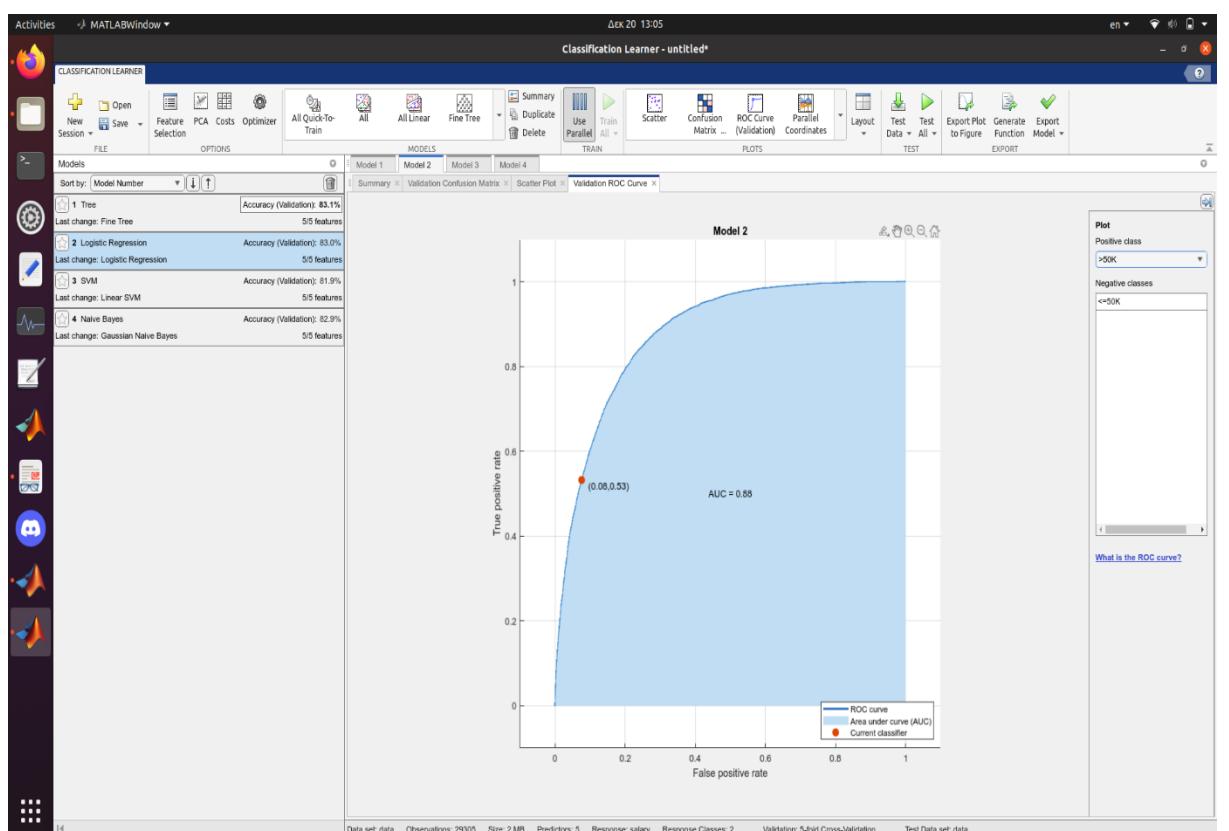
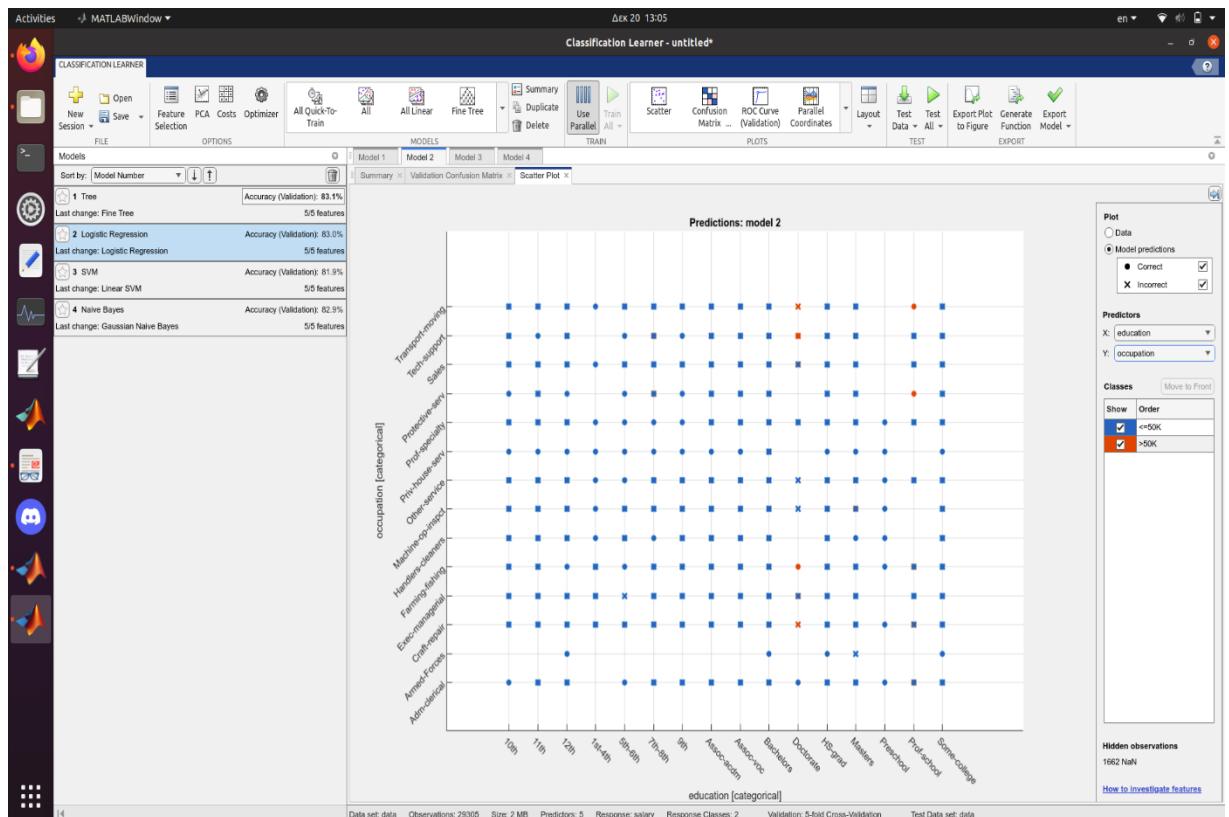
Ακολουθεί ο Πίνακας Σύγχυσης-Confusion Matrix του μοντέλου Fine Tree. Οι σειρές του πίνακα αντιστοιχούν στην αληθινή κλάση, και οι στήλες στην προβλεπόμενη κλάση. Τ διαγώνια κελιά αντιστοιχούν σε σωστά ταξινομημένες παρατηρήσεις. Από τον πίνακα προκύπτουν και οι μετρικές Ακρίβεια, Ανάκληση και F1 Score, που είδαμε και στη θεωρία των μαθήματος. Από την άλλη πλευρά, οι validation ROC curves χρησιμοποιούνται για αξιολόγηση του μοντέλου σε test data.

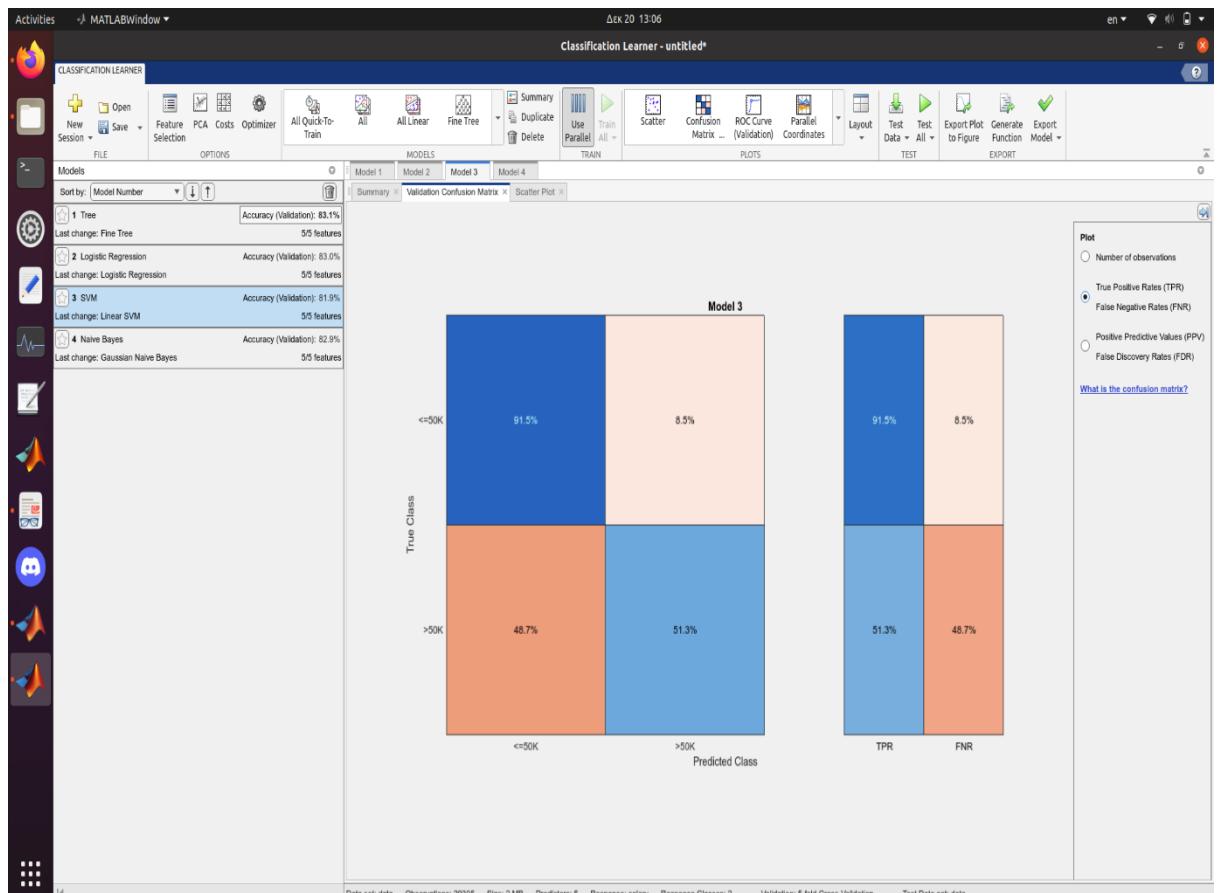
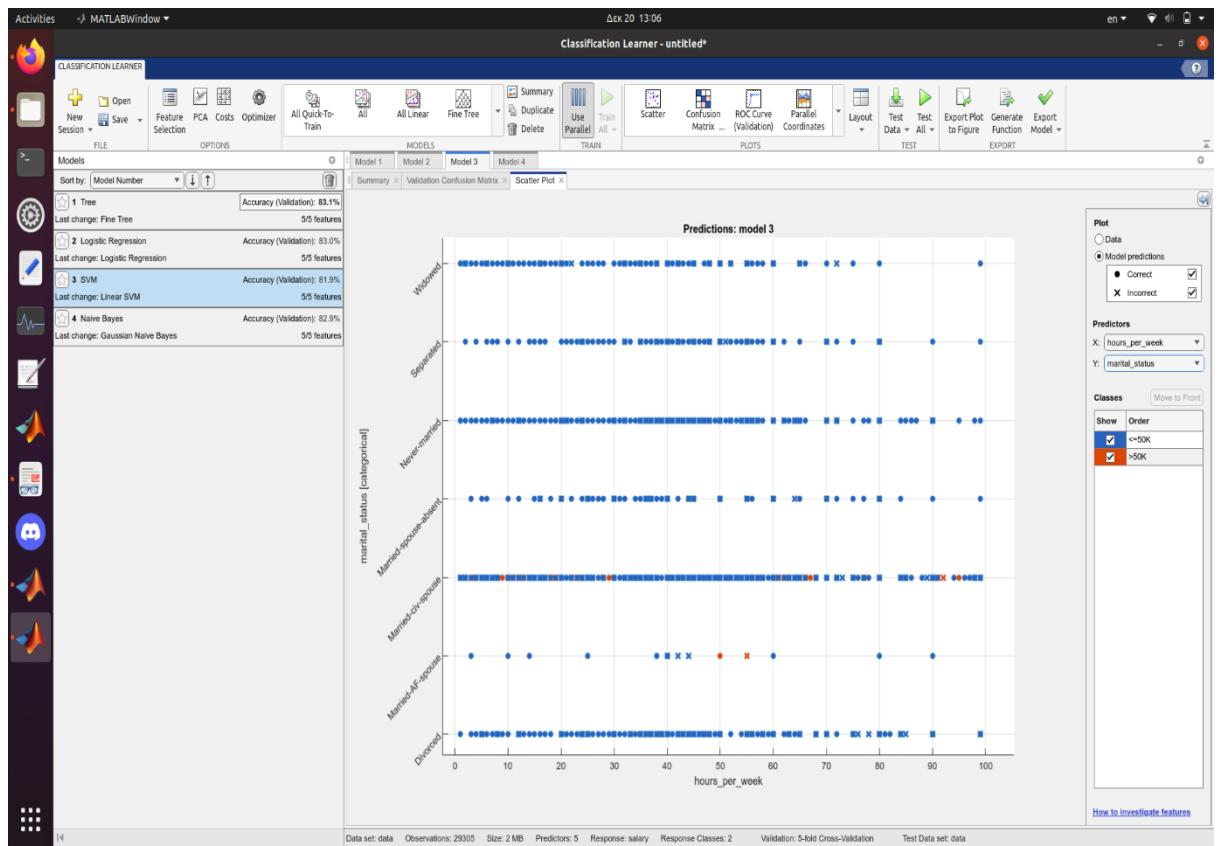


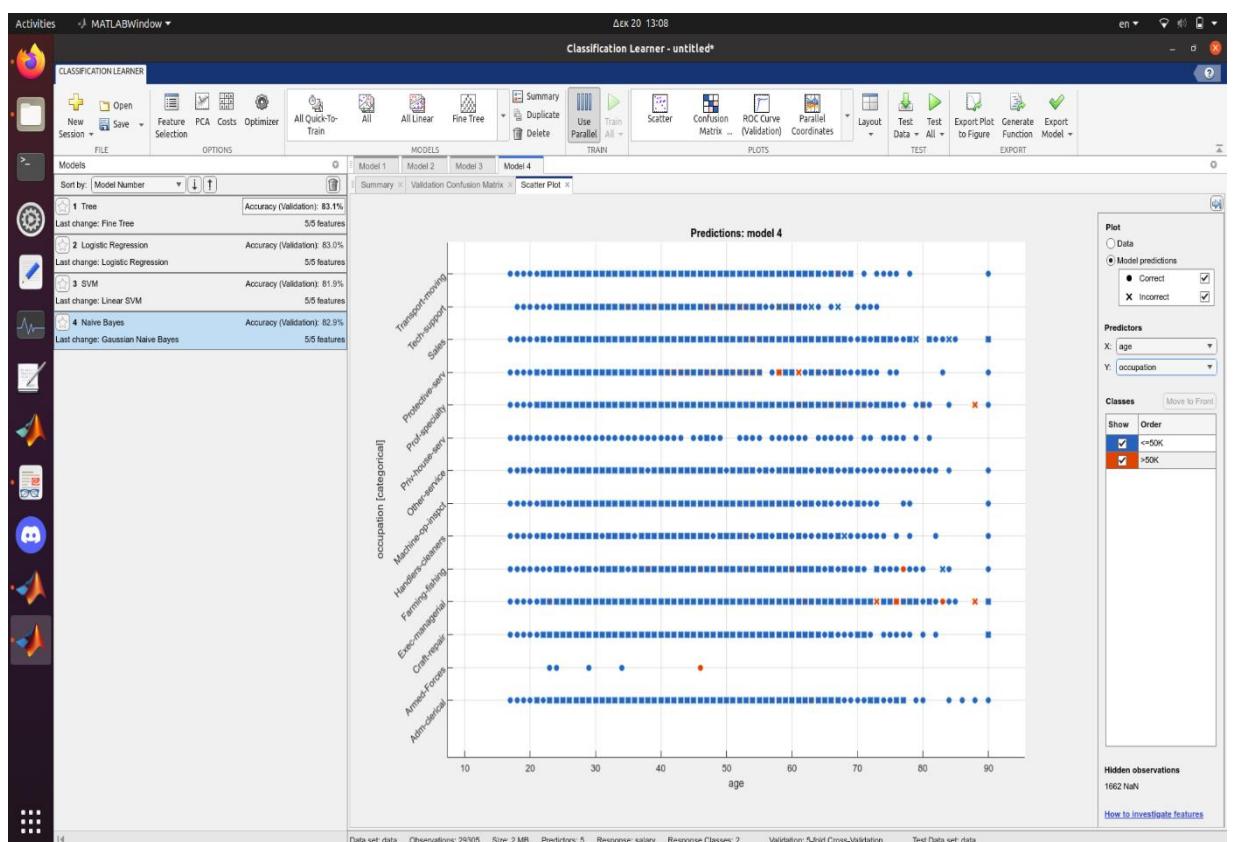
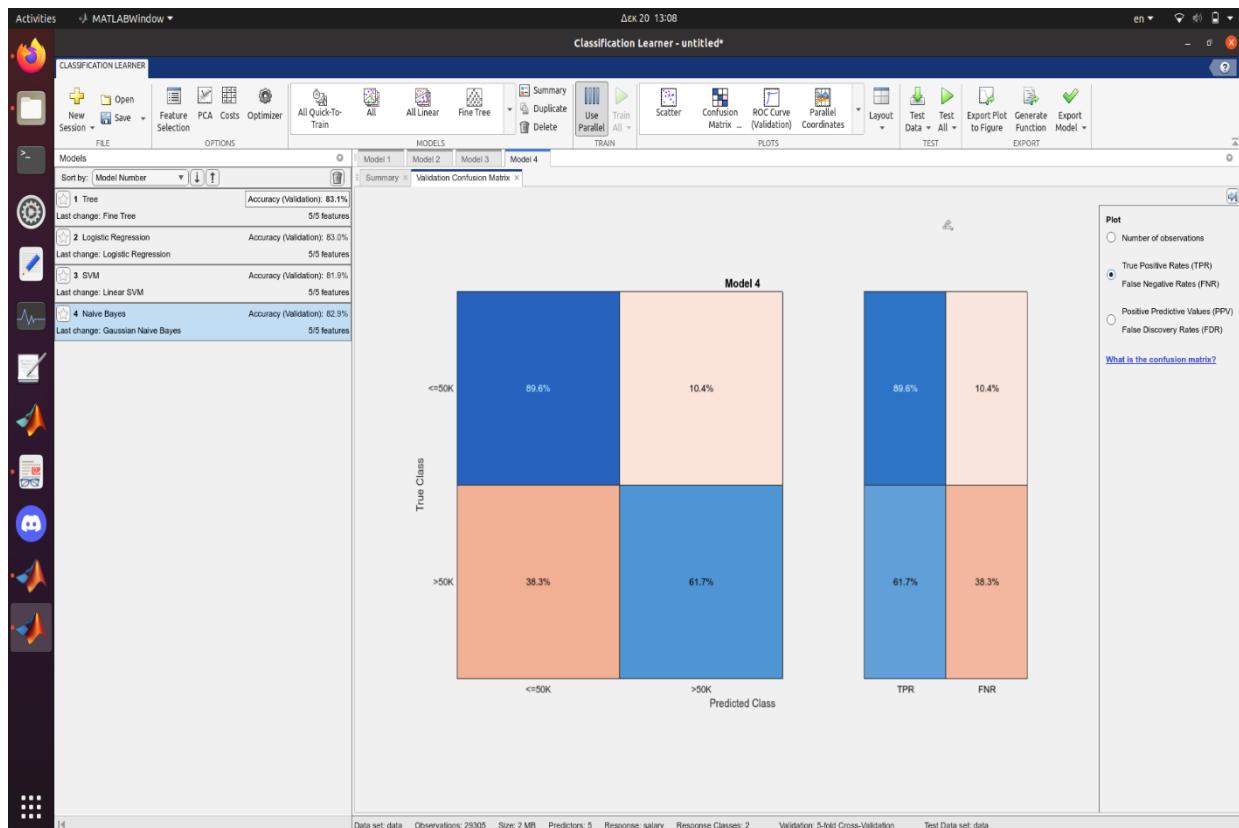


Ακολουθούν διαγράμματα διασποράς, πίνακες σύγχυσης και ROC καμπύλες για τα μοντέλα 2,3 και 4.





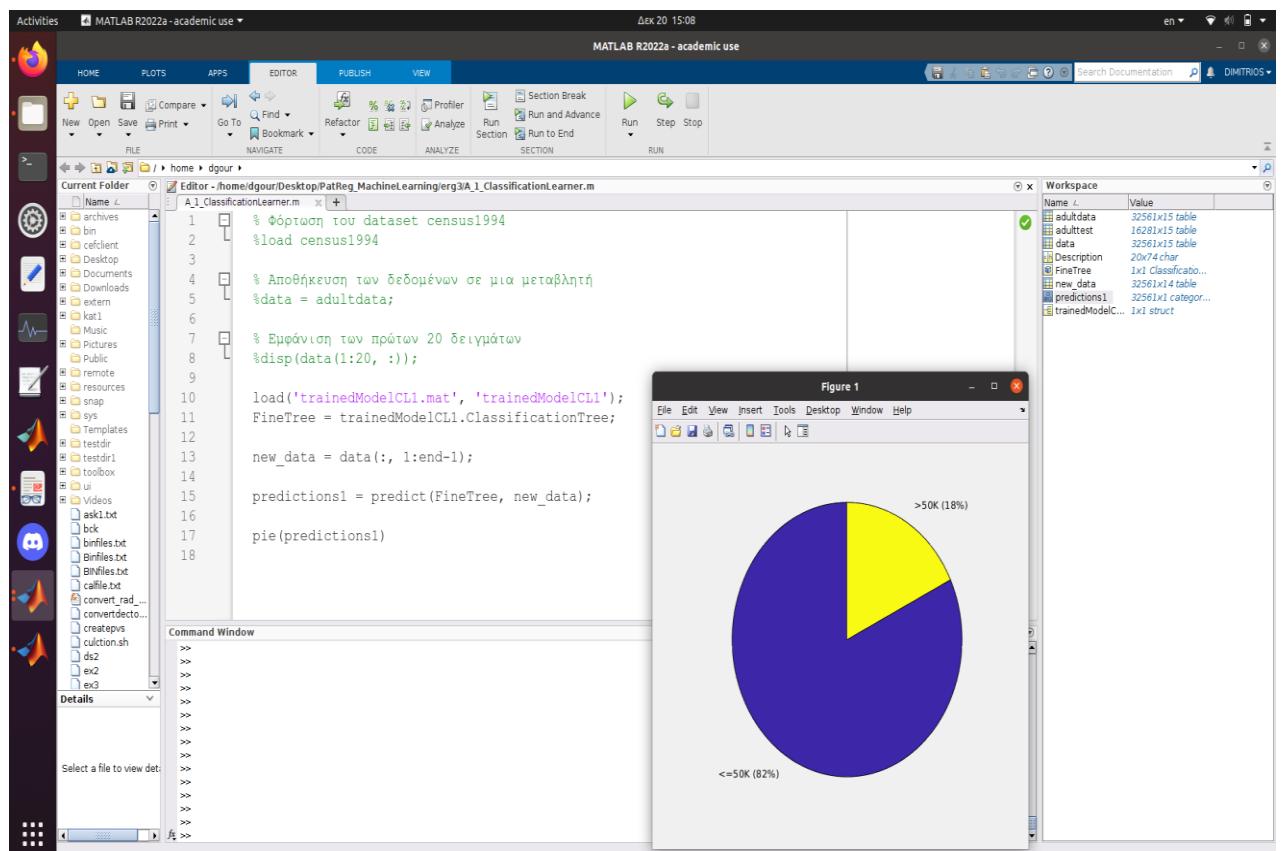
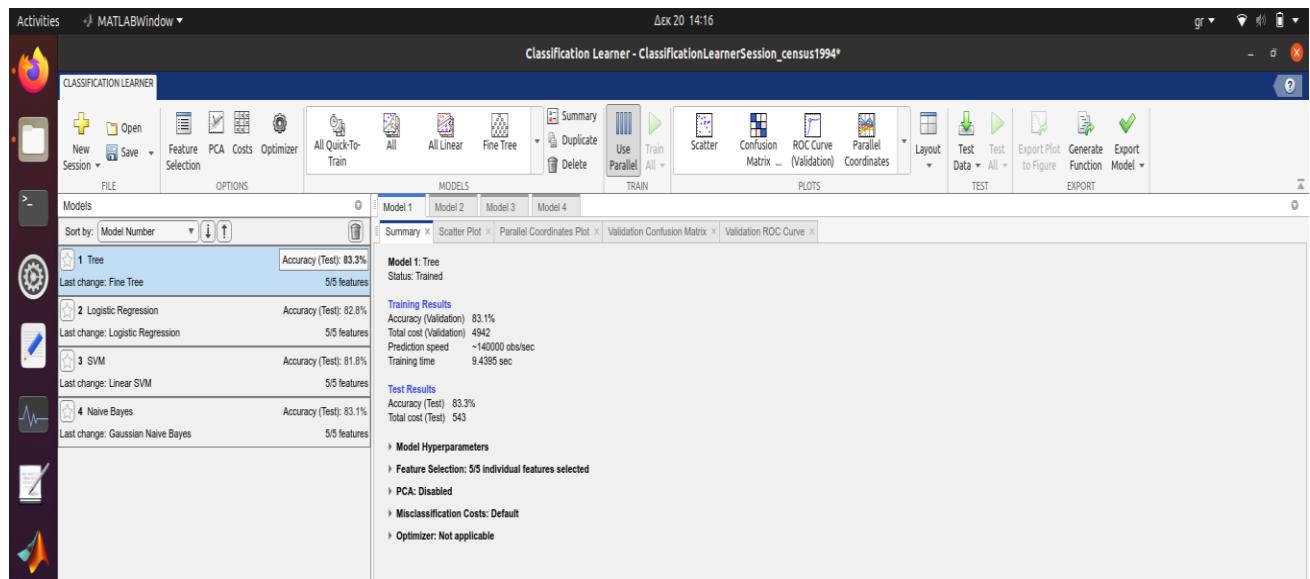


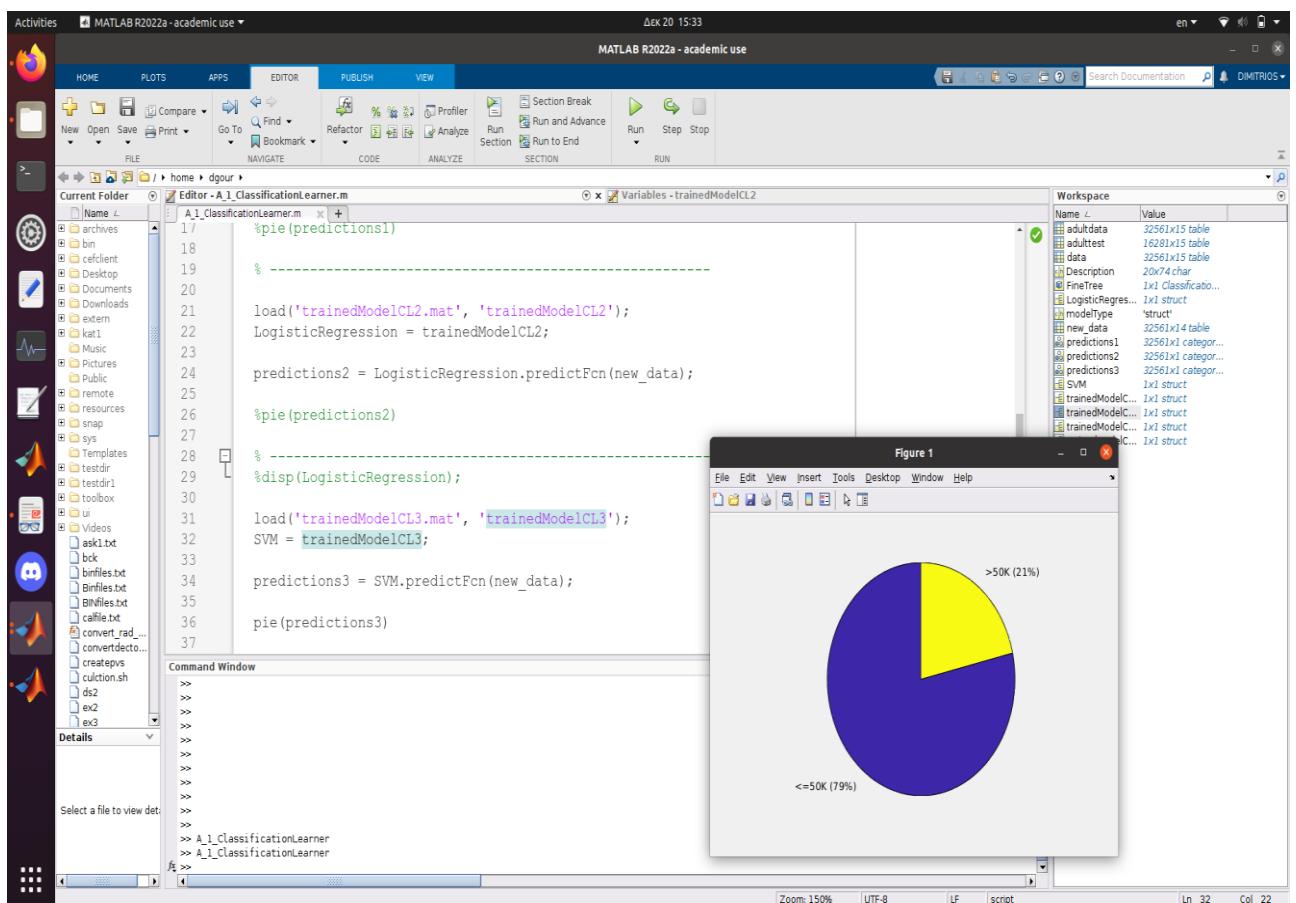
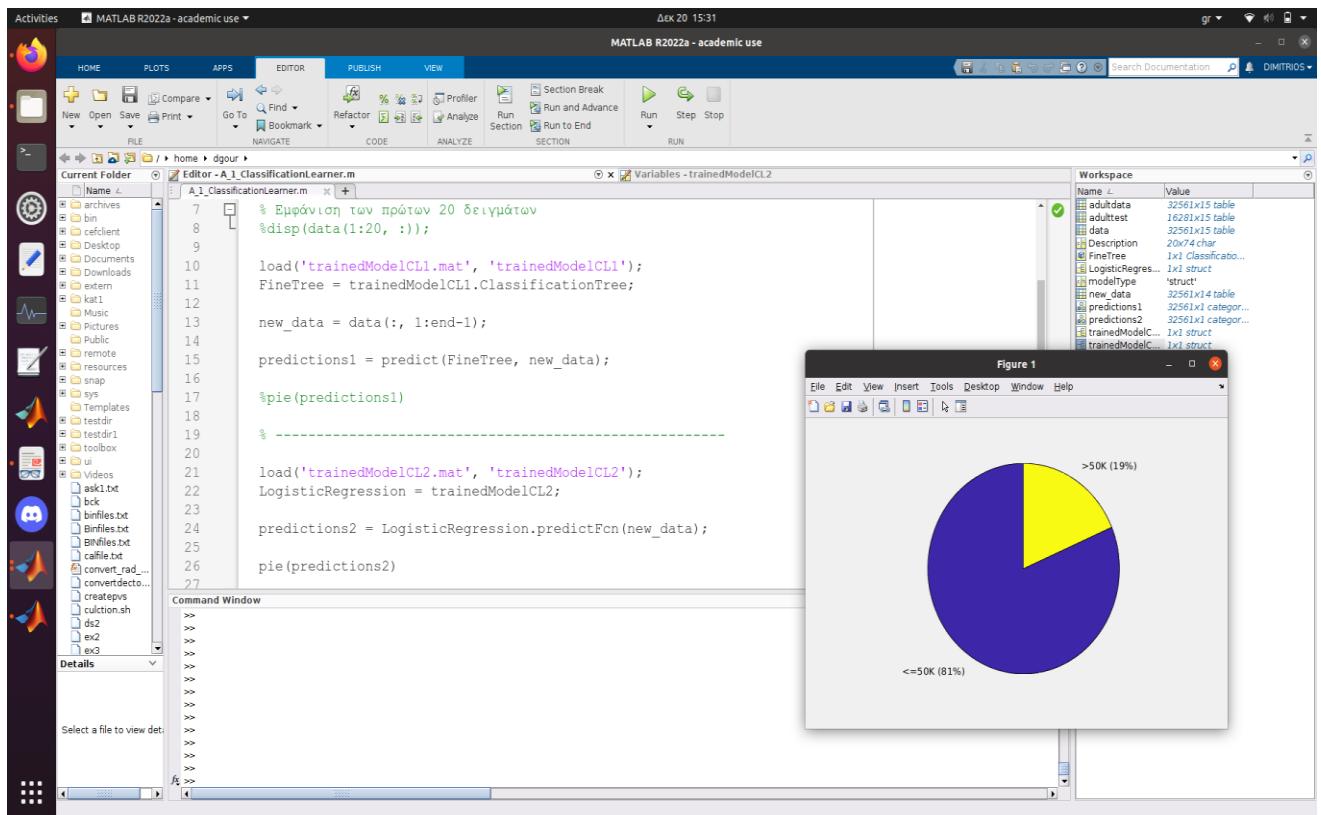


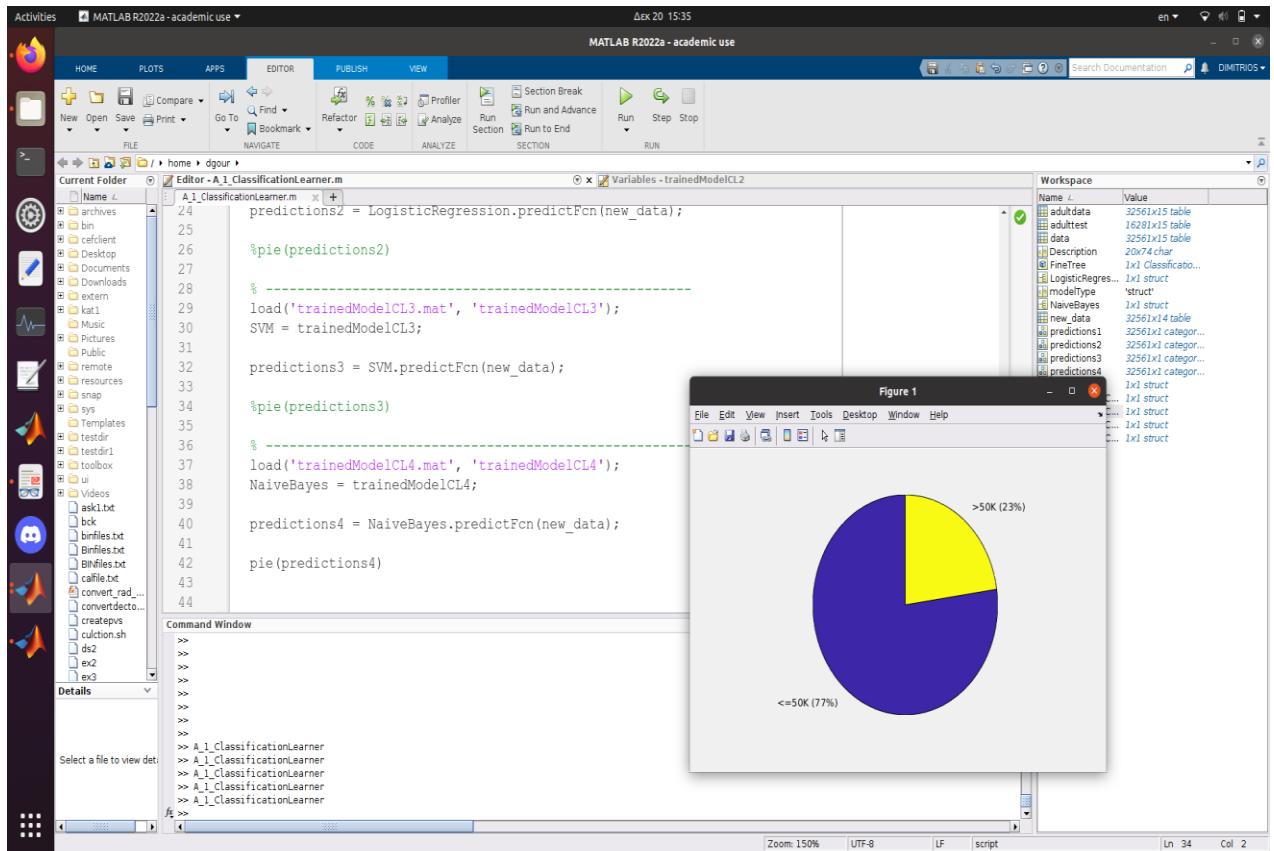
Μέσω των γραφημάτων εκτιμούμε την απόδοση του εκάστοτε μοντέλου. Στο συγκεκριμένο παράδειγμα, το μοντέλο Fine Tree είναι το ποιο αποδοτικό, με μικρή διαφορά από το Logistic Regression.

Ακολουθεί το summary του αποδοτικότερου μοντέλου. Δηλαδή τα training και test results.

Τέλος, απεικονίζονται οι προβλέψεις των 4 μοντέλων με χρήση της μεθόδου predictFcn, πάνω στα new_data.







A2)

1. A_2_RegressionLearner.m

```

url = 'https://archive.ics.uci.edu/ml/machine-learning-
databases/housing/housing.data';
filename = 'housing.data';
outfilename = websave(filename, url);

load(outfilename);

% Προσθήκη ονομάτων στις στήλες
columnNames = {'CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV'};

housing = array2table(housing, 'VariableNames', columnNames);

%CRIM - κατά κεφαλήν ποσοστό εγκληματικότητας ανά πόλη
%ZN - αναλογία γης οικιστικής ζώνης για οικόπεδα ανω των 25.000
τ.μ.
%INDUS - αναλογία στρέμματα επιχειρήσεων μη λιανικής ανά πόλη.
%CHAS - εικονική μεταβλητή του ποταμού Charles (1 εάν η οδός
οριοθετεί το ποτάμι, 0 διαφορετικά)
%NOX - συγκέντρωση νιτρικών οξειδίων (μέρη ανά 10 εκατομμύρια)
%RM - μέσος αριθμός δωματίων ανά κατοικία
%AGE - ποσοστό των ιδιοκατοικούμενων μονάδων που κατασκευάστηκαν
πριν από το 1940
%DIS - σταθμισμένες αποστάσεις σε πέντε κέντρα απασχόλησης της

```

Βοστώνης

%RAD - δείκτης προσβασιμότητας σε ακτινωτούς αυτοκινητόδρομους
%TAX - συντελεστής φόρου ακινήτων πλήρους αξίας ανά **10.000** \$
%PTRATIO - αναλογία μαθητών-δασκάλων ανά πόλη
%LSTAT - % χαμηλότερη κατάσταση του πληθυσμού
%MEDV - Μέση αξία κατοικιών σε **1000** \$

```
% Αφαίρεση της στήλης 'B'  
housing = removevars(housing, 'B');  
  
% Εμφάνιση των πρώτων 5 δειγμάτων με ονόματα στις στήλες  
disp(housing(1:5, :))  
  
load('trainedModelRL1.mat', 'trainedModelRL1');  
FineTree = trainedModelRL1.RegessionTree;  
  
new_data = housing(:, 1:end-1);  
  
predictions1 = predict(FineTree, new_data);  
  
% -----  
  
load('trainedModelRL2.mat', 'trainedModelRL2');  
LogisticRegression = trainedModelRL2;  
  
predictions2 = LogisticRegression.predictFcn(new_data);  
  
% -----  
load('trainedModelRL3.mat', 'trainedModelRL3');  
SVM = trainedModelRL3;  
  
predictions3 = SVM.predictFcn(new_data);  
  
% -----  
load('trainedModelRL4.mat', 'trainedModelRL4');  
LinearRegression_LinearInteractions = trainedModelRL4;  
  
predictions4 =  
LinearRegression_LinearInteractions.predictFcn(new_data);  
  
% -----  
% -----  
  
figure;  
  
bar([predictions1, predictions2, predictions3, predictions4],  
'stacked');  
title('Σύγκριση Προβλέψεων Μοντέλων');  
xlabel('Δείγματα');  
ylabel('Προβλέψεις');  
legend('Μοντέλο 1', 'Μοντέλο 2', 'Μοντέλο 3', 'Μοντέλο 4');
```

2. Μοντέλα:

Fine Tree

Linear Regression

Linear SVM

Linear Regression – Interactions Linear

3. Χρόνος εκπαίδευσης: 5.9,6.2,5.0,6.2

Accuracy-Validation(σε RMSE): 4.3,4.8,5.0,3.3

RMSE(Root Mean Squared Error): Μέτρο αξιολόγησης για την απόδοση ενός Regression Model. Όσο μικρότερο είναι, τόσο καλύτερη η απόδοση του μοντέλου.

Accuracy-Test(σε RMSE): 2.6,5.3,5.5,4.0

4. Export των μοντέλων σε Workspace

Σύντομη περιγραφή του προβλήματος

To Boston housing dataset είναι ιδανικό για προβλήματα παλινδρόμησης. Χρησιμοποιούμε το App:Regression Learner. Το dataset περιέχει δεδομένα που σχετίζονται με κατοικίες και έχει ως στόχο να προβλέψει μια συνεχή μεταβλητή, δηλαδή τη μέση τιμή κατοικίας. Τα δεδομένα συλλέχθηκαν από βάση δεδομένων, που αναγράφεται στον κώδικα ως URL.

Στο dataset περιλαμβάνονται τα εξής χαρακτηριστικά :εγκληματικότητα, εικονική μεταβλητή του ποταμού Charles, συντελεστής φόρου ακινήτου, δείκτης προσβασιμότητας σε αυτοκινητόδρομους και αρκετά άλλα.

Αξίζει να σημειωθεί ότι το dataset αφορά δεδομένα παλαιότερης δεκαετίας.

Παρατηρήσεις

Τα βήματα που ακολουθήσαμε για την επεξεργασία του dataset ακολουθούν τα βήματα του 1^{ου} dataset. Οι διαφοροποιήσεις που προέκυψαν θα παρουσιαστούν σε αυτό το σημείο.

Ως predictors επιλέχθηκαν όλα τα χαρακτηριστικά. Παρουσιάζονται διαγράμματα διασποράς του μοντέλου Fine Tree με τα εξής x-axis:CRIM,RM,TAX,CHAS. Επίσης, δημιουργήθηκε και box plot με x-axis το CHAS.

Επίσης, παρουσιάζονται διαγράμματα διασποράς, όπου σε κοινό γράφημα υπάρχουν True και Predicted δεδομένα(με μπλε και κίτρινους κύκλους αντίστοιχα). Επιπροσθέτως, υπάρχει το διάγραμμα Validation Predicted vs Actual Plot, που βλέπουμε και το Perfect Prediction. Ακόμη, παρουσιάζουμε το Validation Residuals Plot, για έλεγχο της απόδοσης του μοντέλου.

Συνοπτικά, εμφανίζει τη διαφορά μεταξύ των προβλεπόμενων και των αληθινών αποκρίσεων. Στον άξονα x χρησιμοποιούμε τον predictor CRIM ή την predicted response.

Επιπροσθέτως, μέσω του Layout->Compare Models συγκρίνουμε διαγράμματα διασποράς των μοντέλων Linear Regression και Linear SVM. Ακολουθώντας την ίδια τακτική συγκρίνουμε τα Validation Residuals Plots των μοντέλων μεταξύ τους.

Επιπλέον, εμφανίζονται τα summary της εκπαίδευσης και του testing όλων των μοντέλων.

Γίνεται χρήση του γραφήματος Feature Importance Scores sorted using F Test algorithm. Με λίγα λόγια, απεικονίζει τη σημασία των χαρακτηριστικών στο μοντέλο, ταξινομημένη με βάση τον αλγόριθμο F Test, για τον υπολογισμό του F-score. Μετά από περιήγηση στη σελίδα του Matlab, καταλήξαμε στο συμπέρασμα ότι χαρακτηριστικά με υψηλότερο F-score θεωρούνται πιο σημαντικά για το μοντέλο. Το συγκεκριμένο γράφημα έχει τη δυνατότητα να μας κατευθύνει στην επιλογή συγκεκριμένων χαρακτηριστικών κατά την προεπεξεργασία των δεδομένων.

Τέλος, με κώδικα εκτιμάμε τις προβλέψεις των 4 μοντέλων. Έπειτα, παρουσιάζουμε τις παραπάνω προβλέψεις σε ένα κοινό γράφημα, με τη βοήθεια της συνάρτησης bar. Ο άξονας x αποτελείται από τον αριθμό των δειγμάτων, ενώ ο άξονας y από τις προβλέψεις. Η πρόβλεψη κάθε μοντέλου έχει άλλο χρώμα. Παρατηρούμε πως το 4^ο μοντέλο, Linear Regression – Interactions Linear, κάνει τις σωστότερες προβλέψεις. Το συγκεκριμένο γεγονός επαληθεύεται και από το RMSE.

Activities MATLABWindow ▾ ΔEK 21 12:19

MATLAB R2022a - academic use

Regression Learner

FILE PLOTS APPS EDITOR PUBLISH VIEW

REGRESSION LEARNER

New Session Save Feature Selection All Quick-To-Train All Models Summary Duplicate Train Parallel Response Predicted vs. Actual Residuals Predicted vs. Actual (Test) Layout Test Data All TEST EXPORT

Models Sort by: Model Number

Data set

Data Set Variable: housing 506x13 table

Response: MEDV

Predictors:

Name	Type	Range
CRIM	double	0.0632..88.972
ZN	double	0..100
INDUS	double	0.46..27.74
CHAS	double	0..1
NOX	double	0.365..0.871
RM	double	3.561..8.78

Add All Remove All

Validation

Validation Scheme: Cross-Validation

Cross-validation folds: 5

Read about validation

Test

Set aside a test data set: checked

Percent set aside: 10

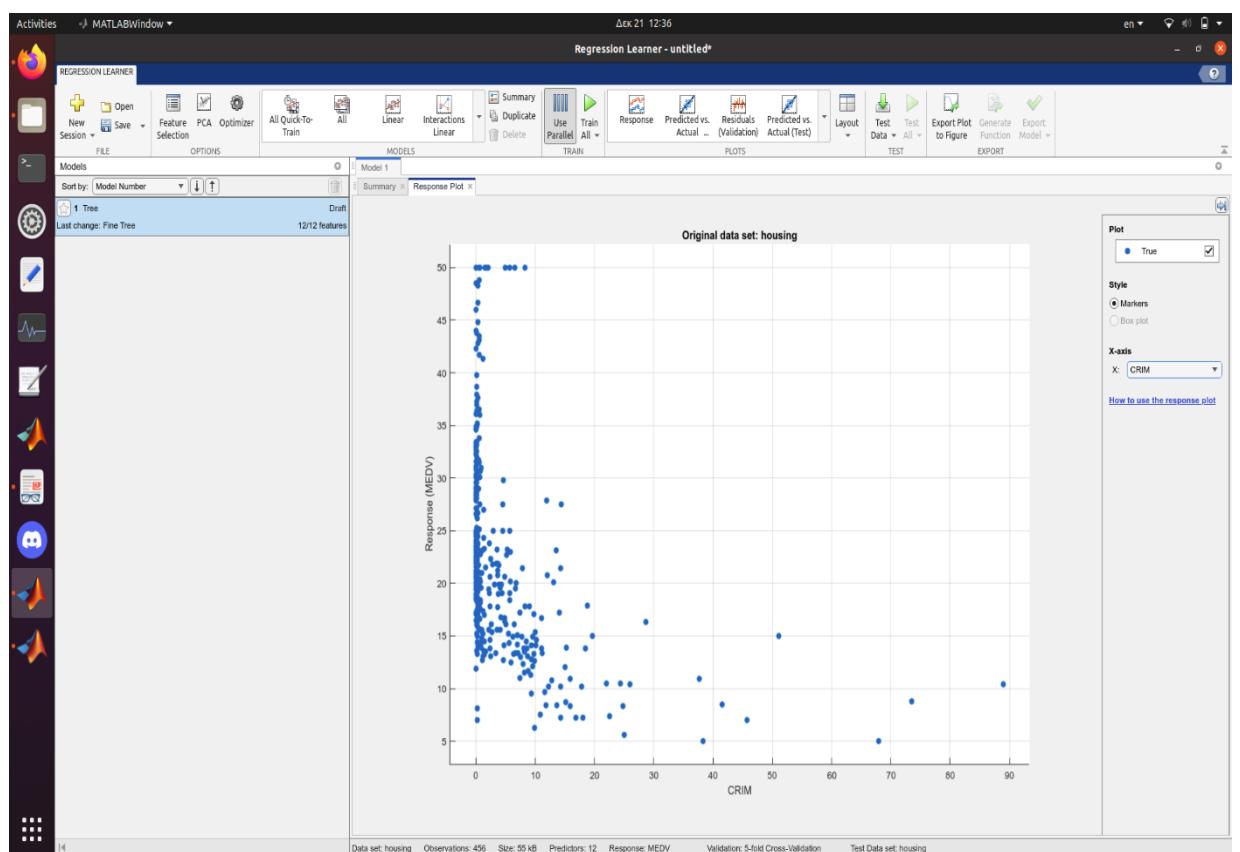
Read about test data

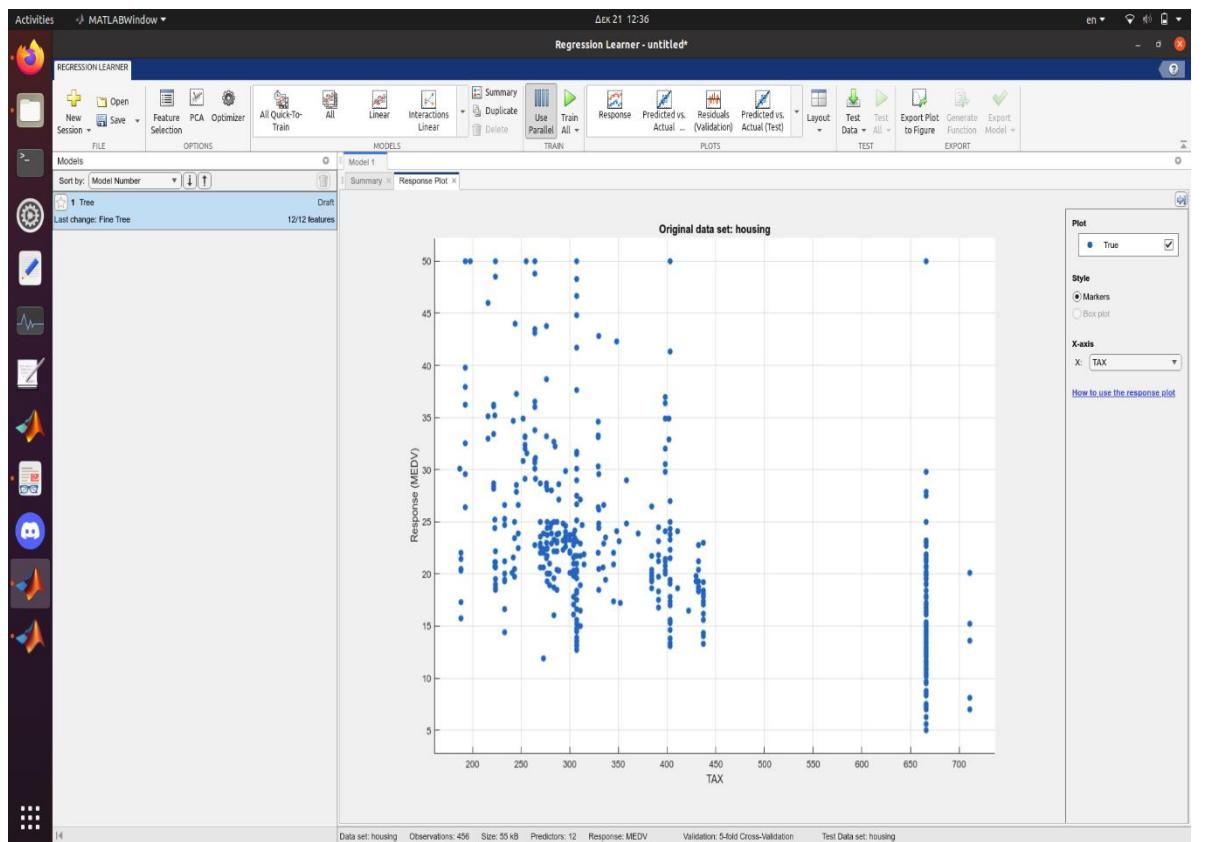
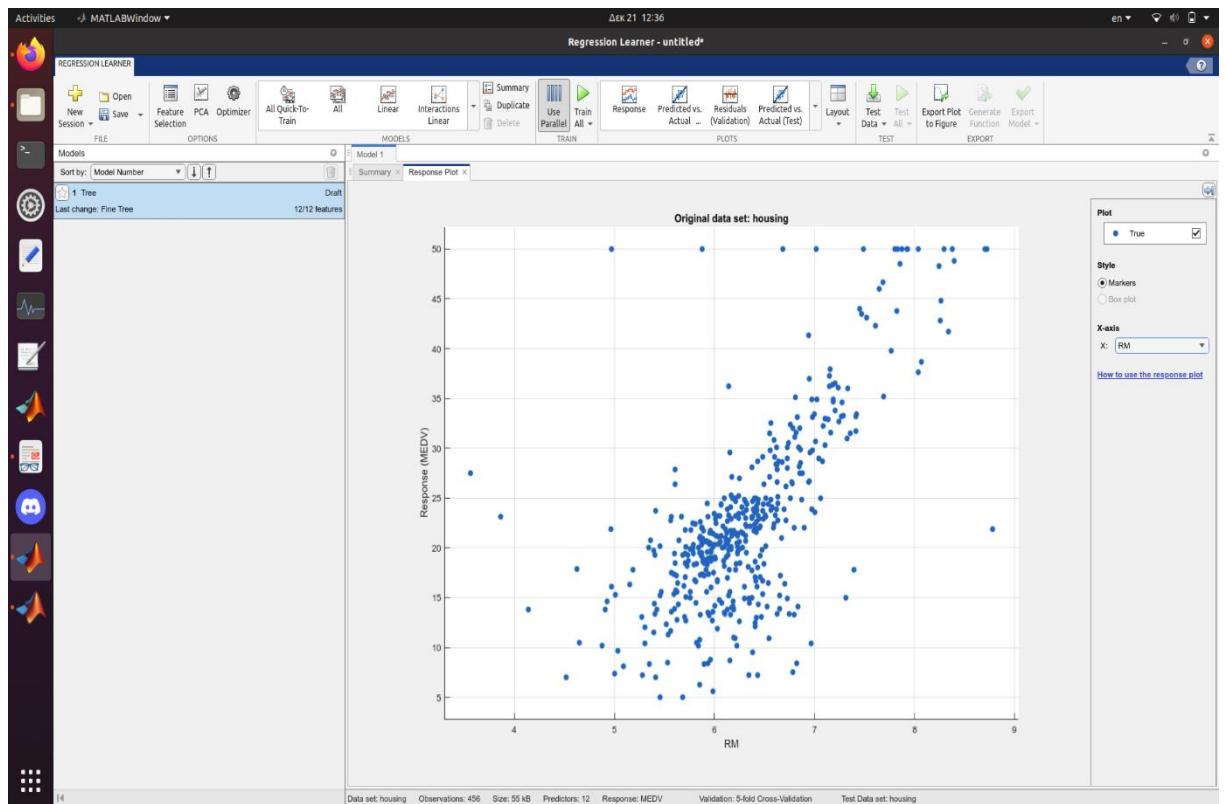
Start Session Cancel

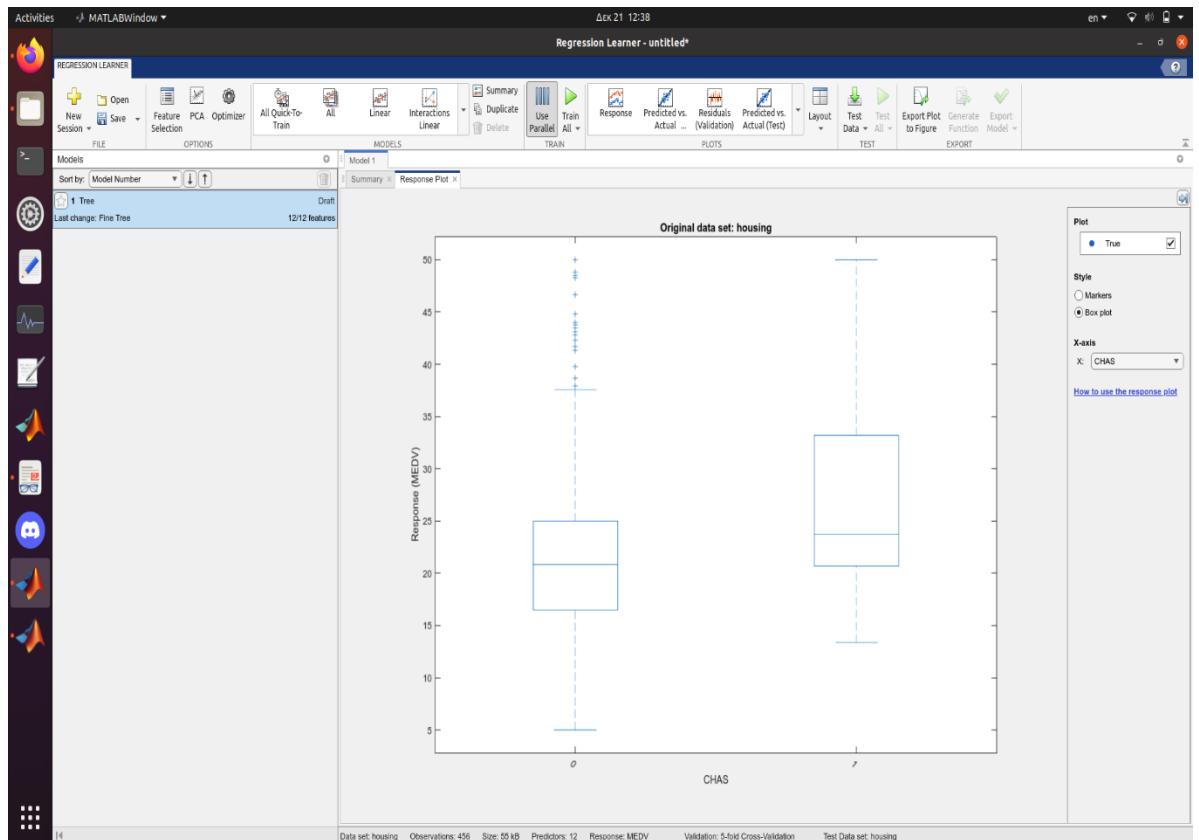
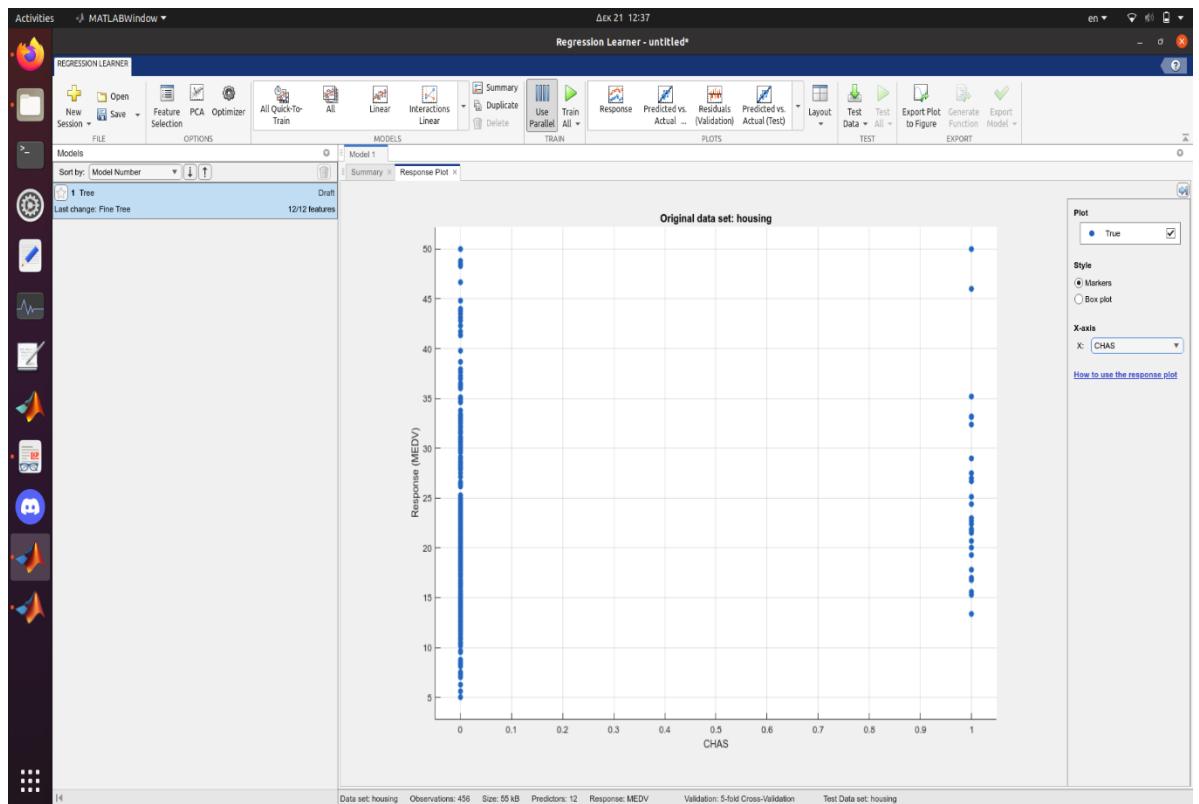
No details available

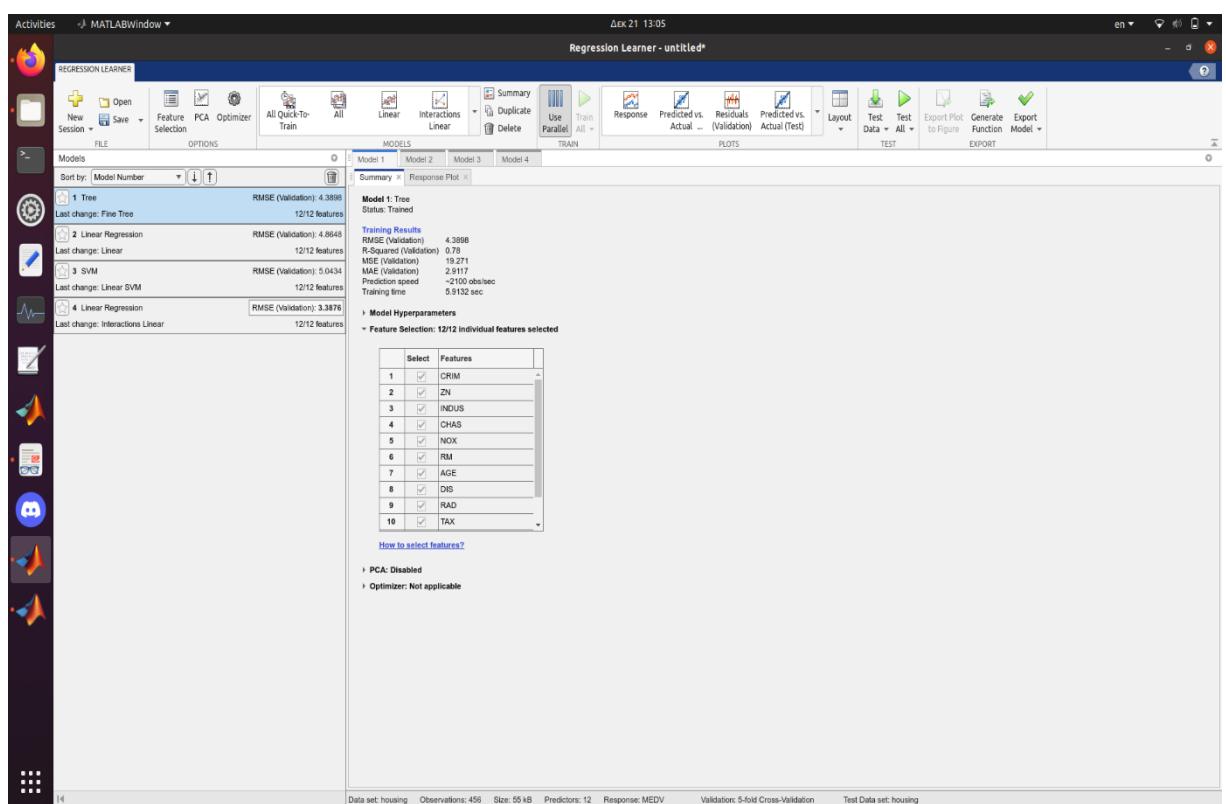
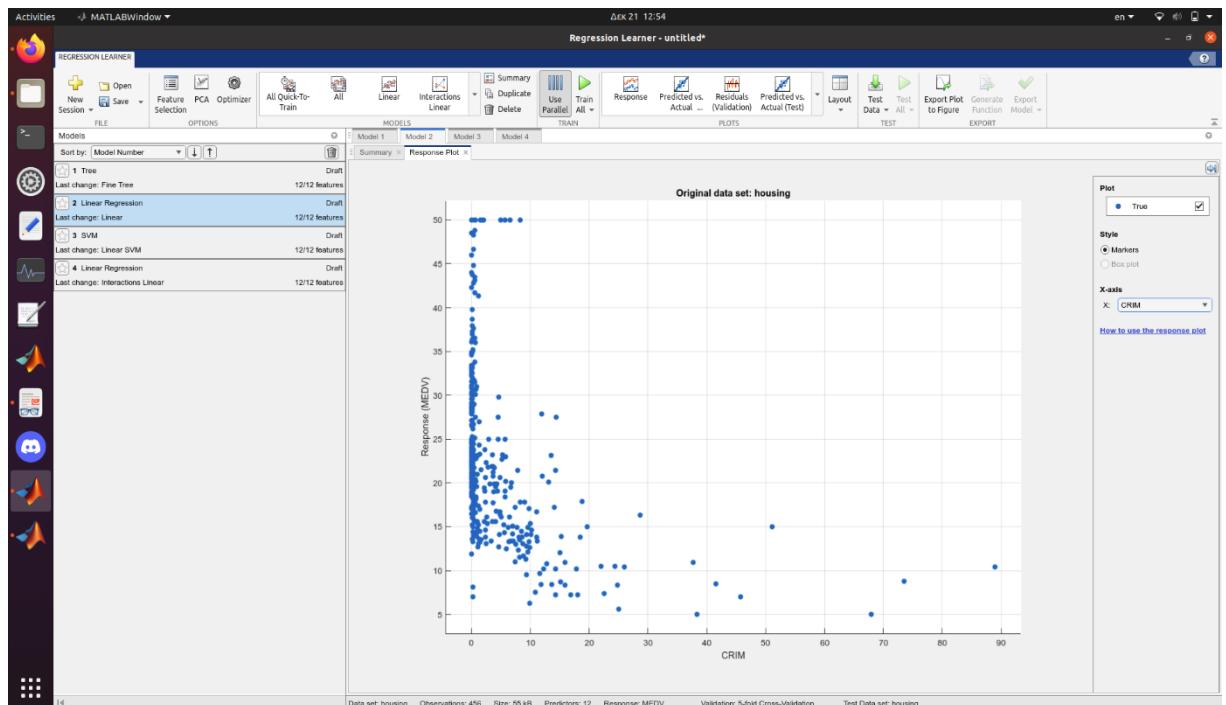
f2 >

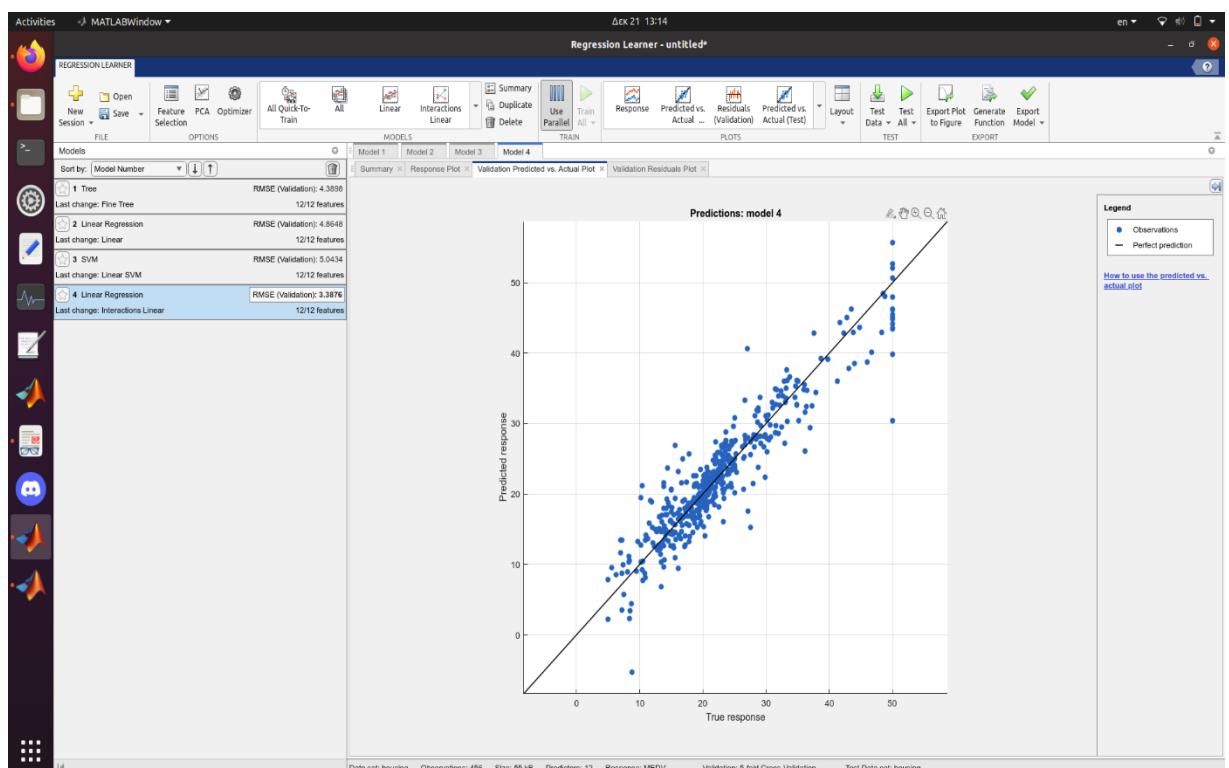
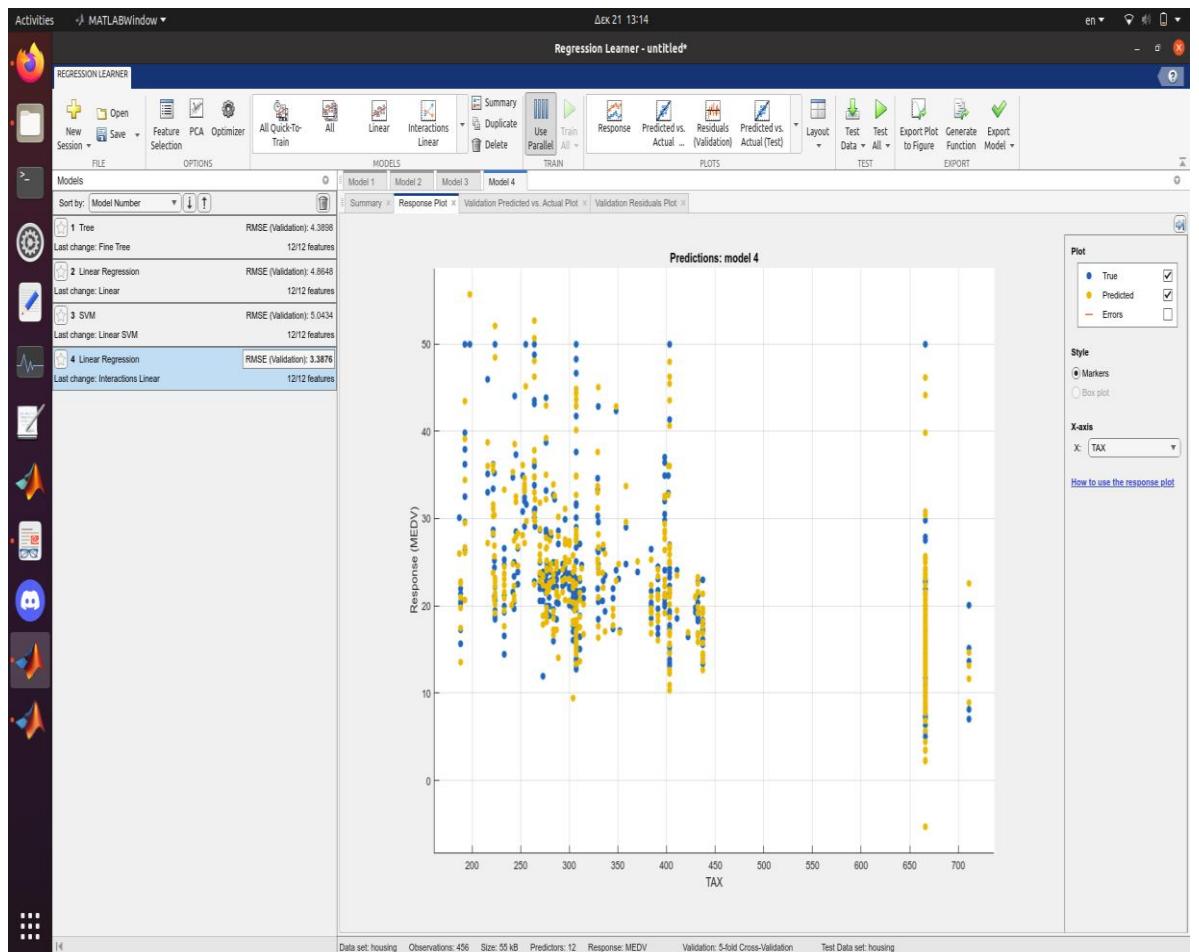
Zoom: 150% UFT-8 LF script Ln 30 Col 1

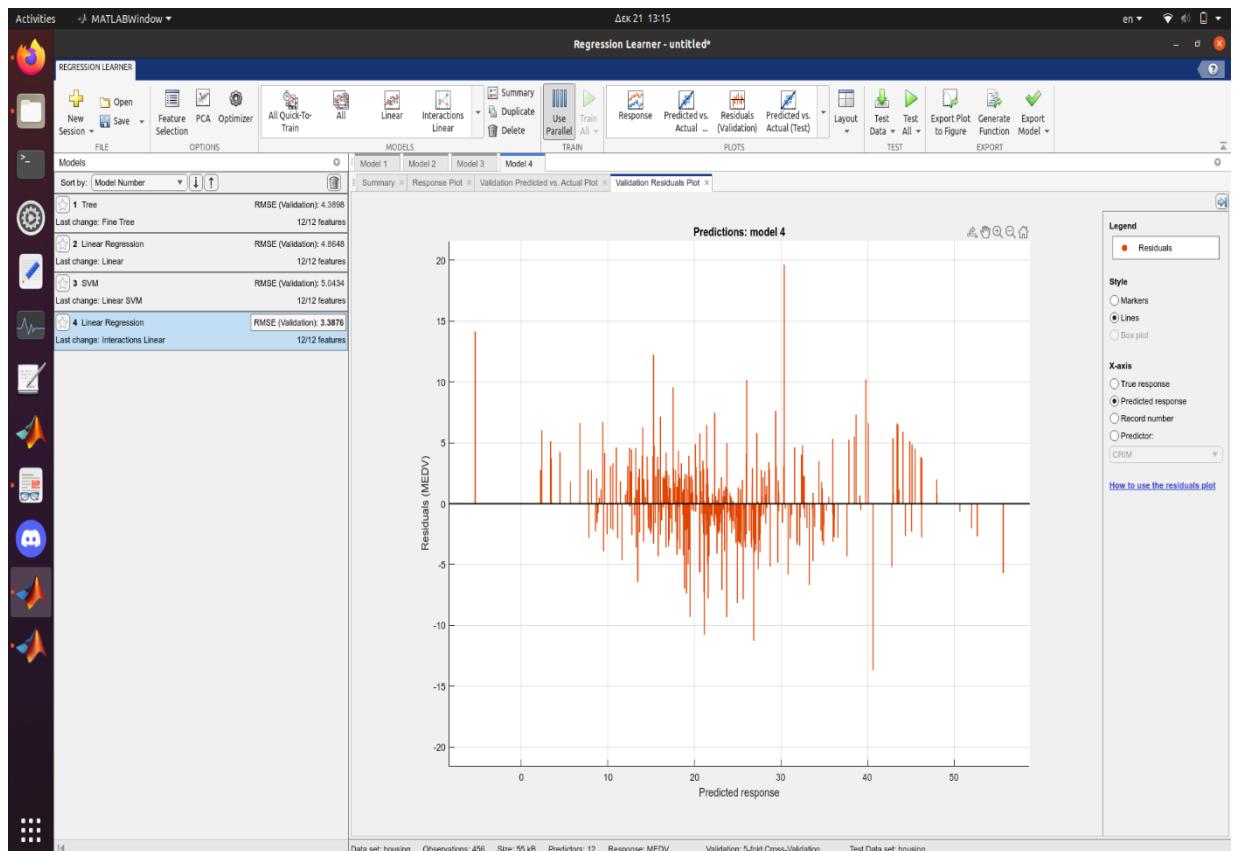
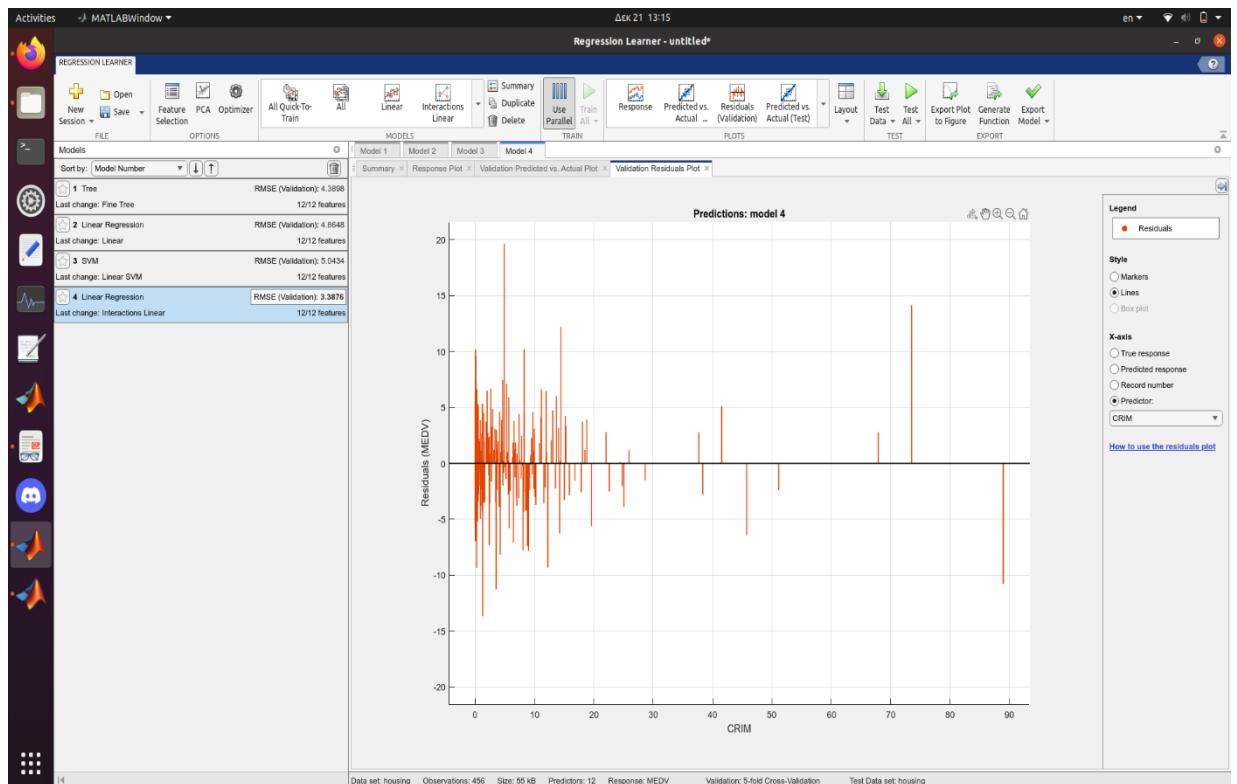


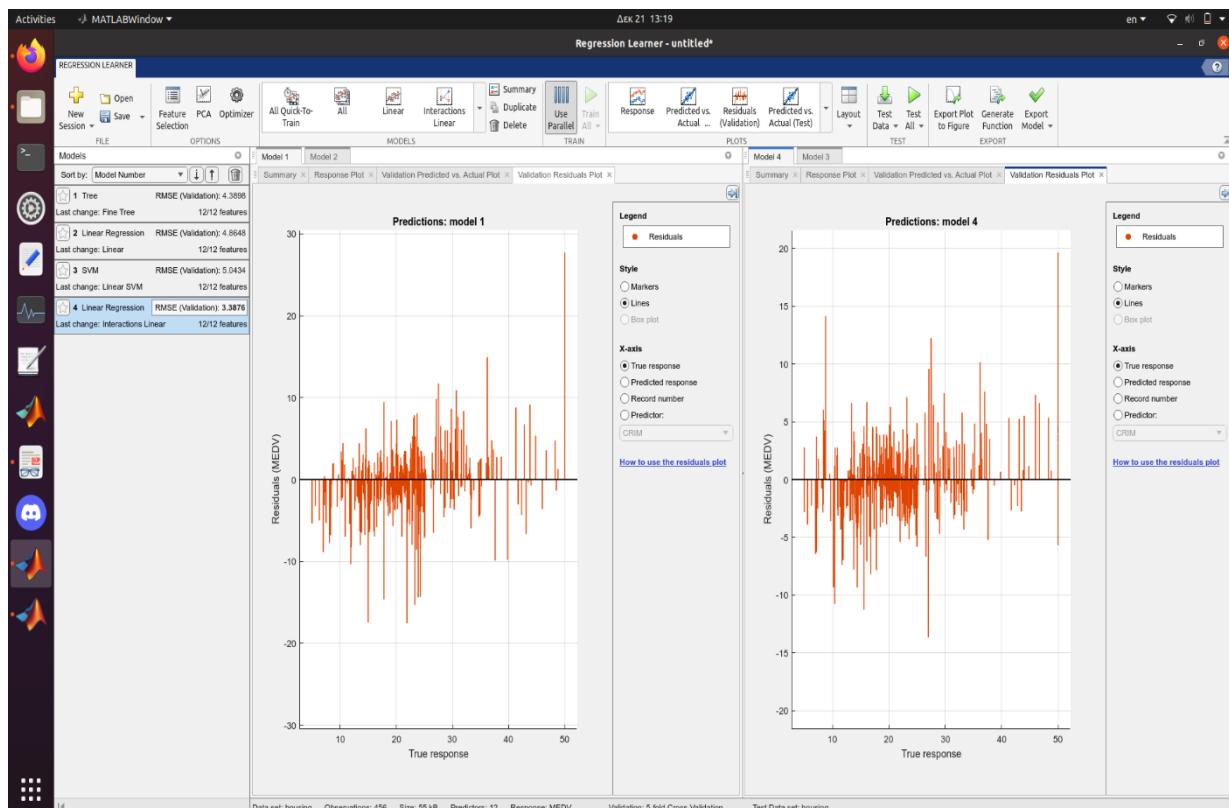
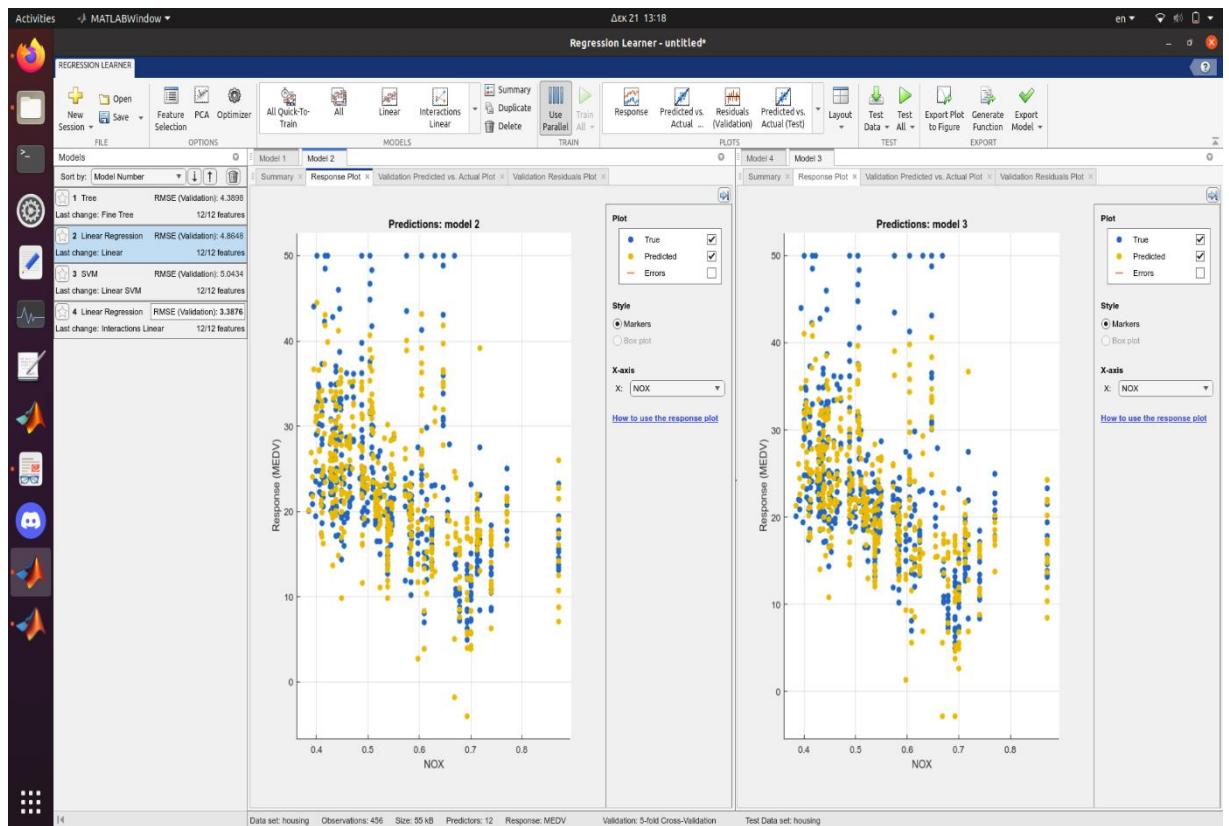


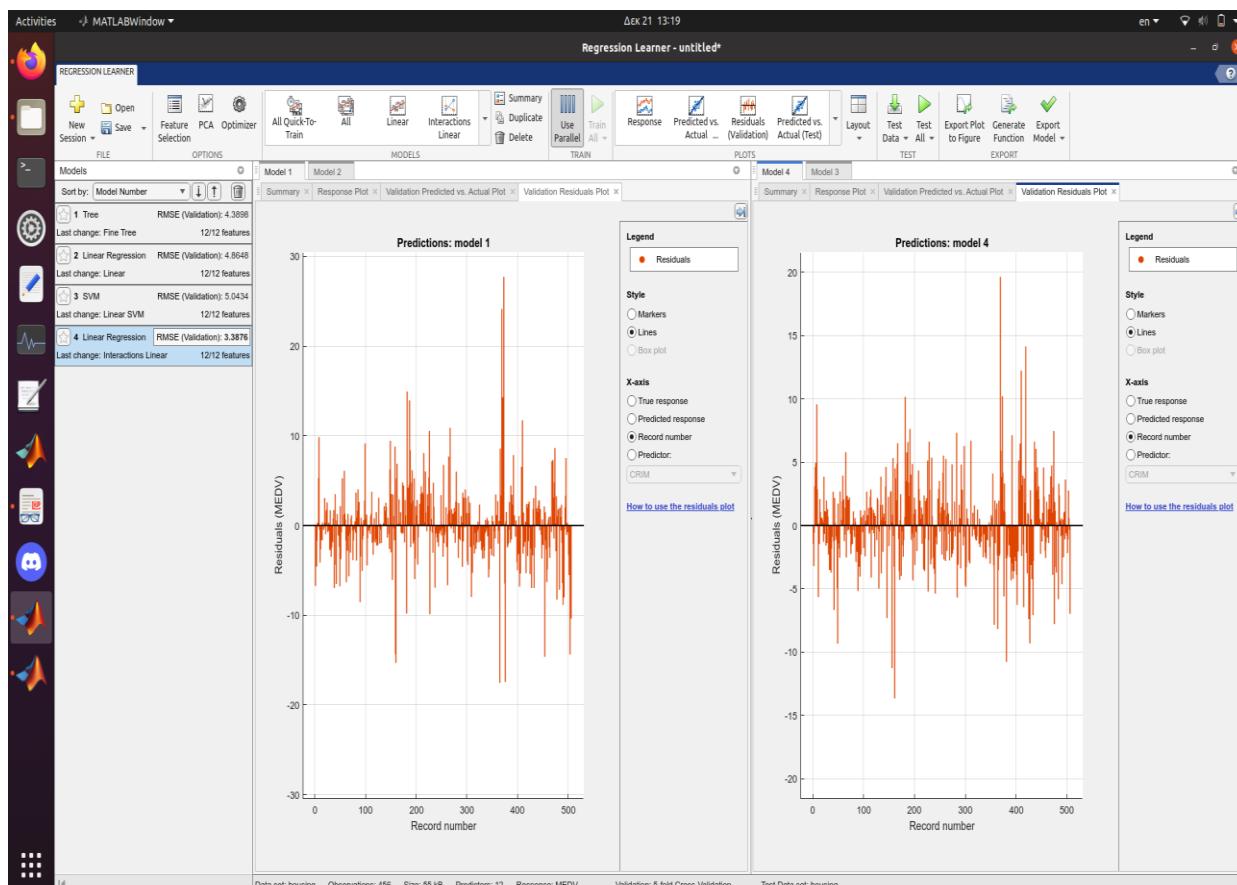
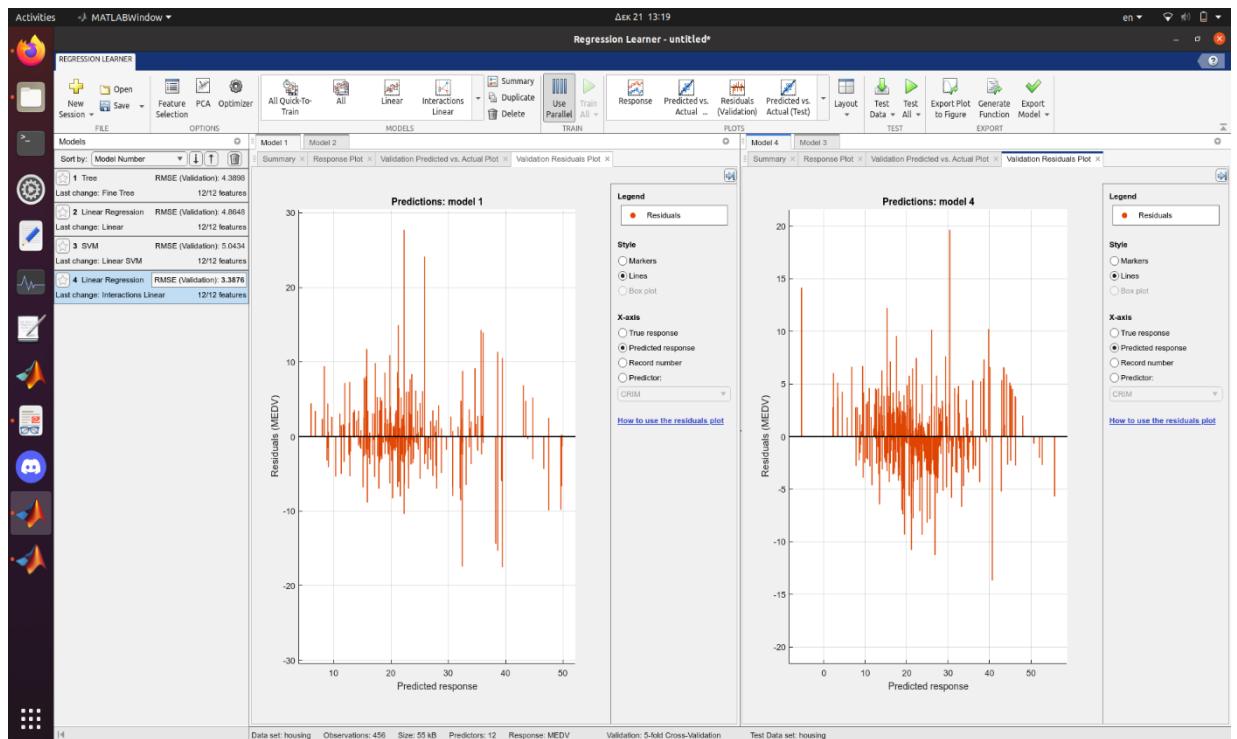


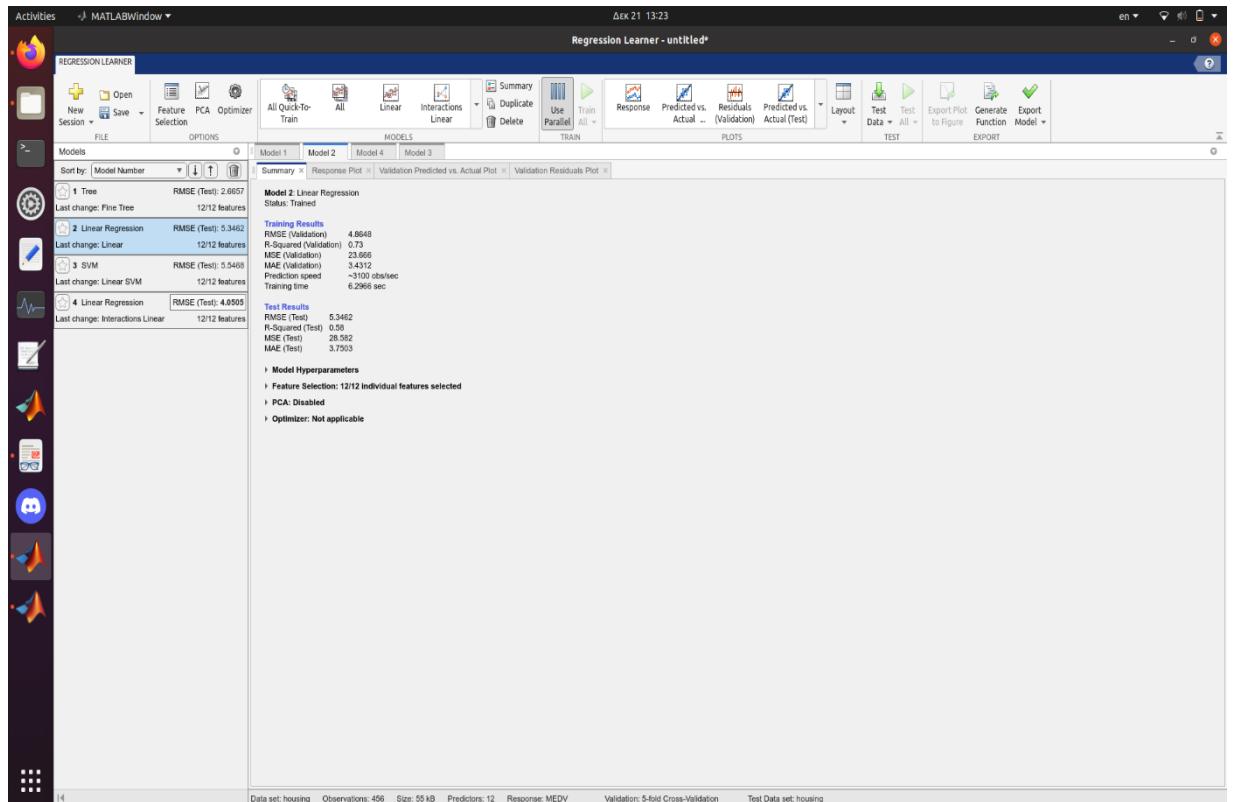
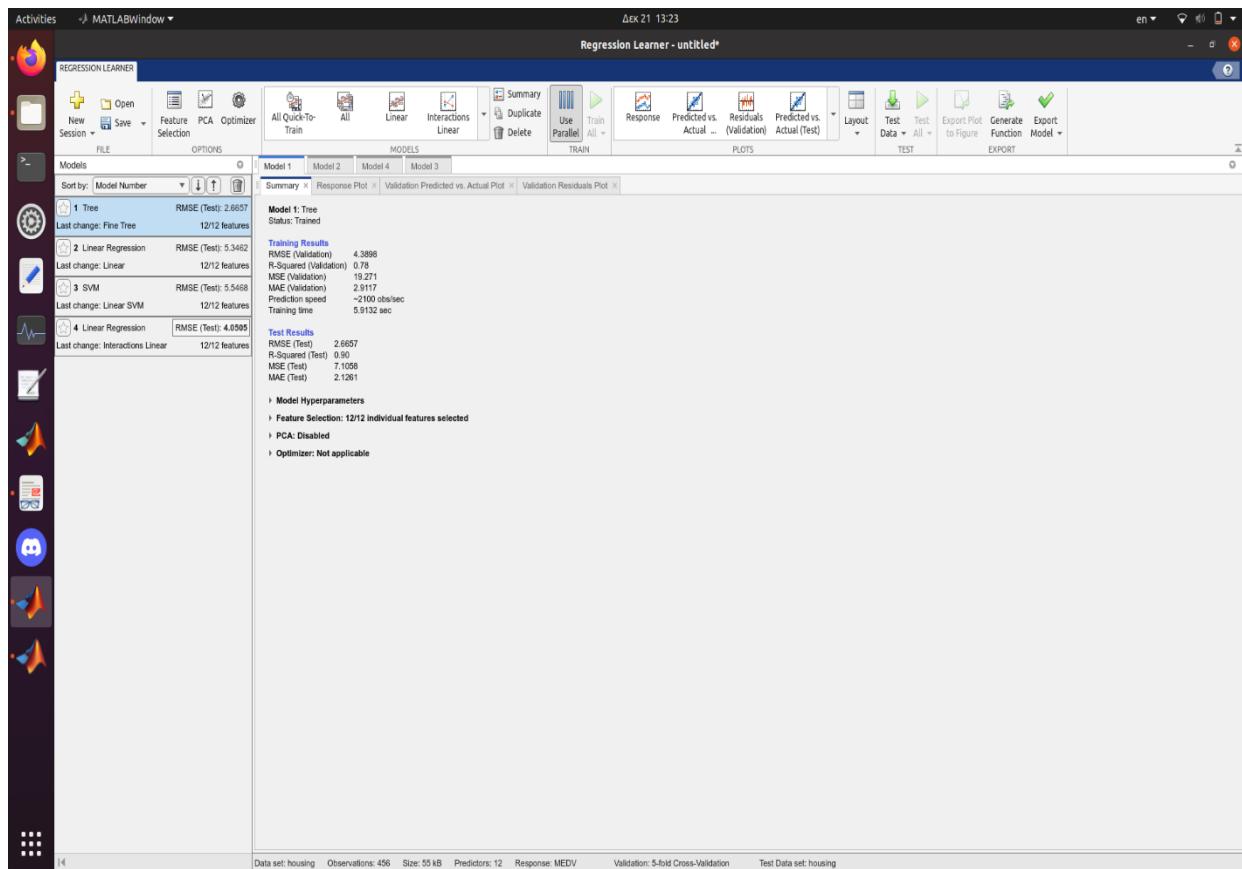


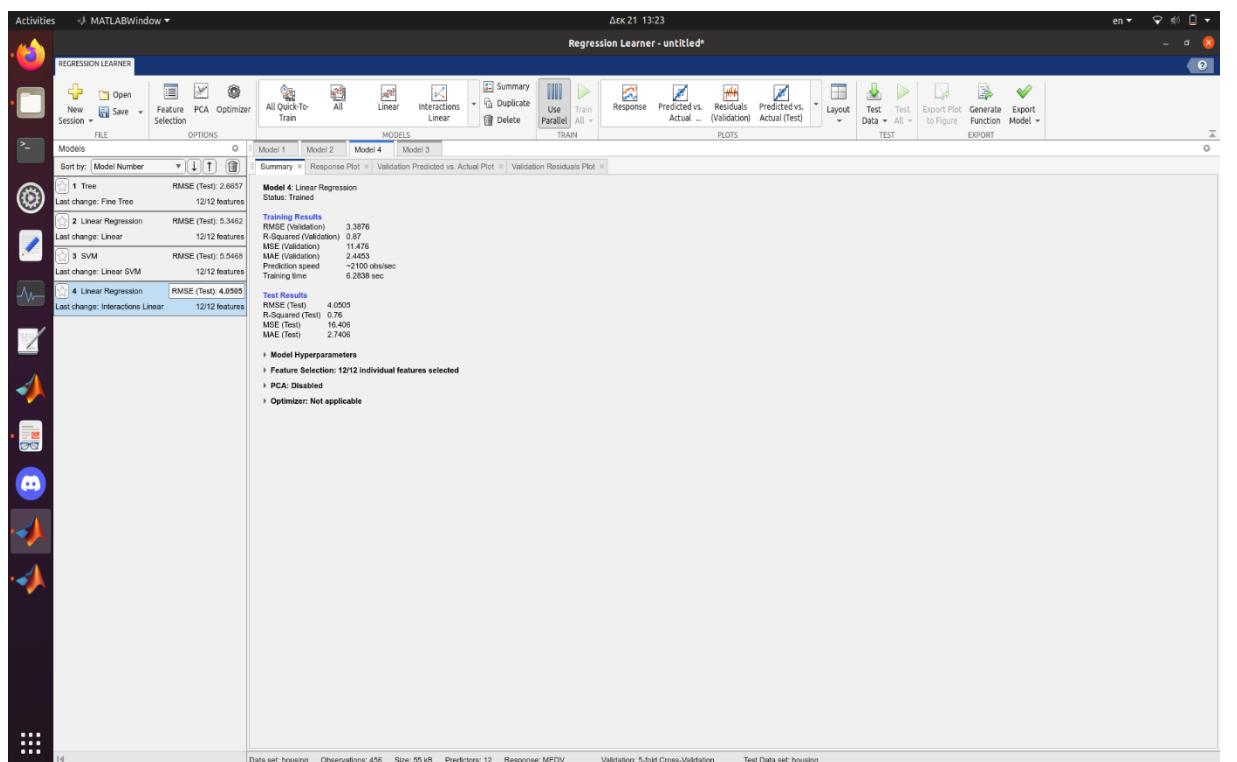
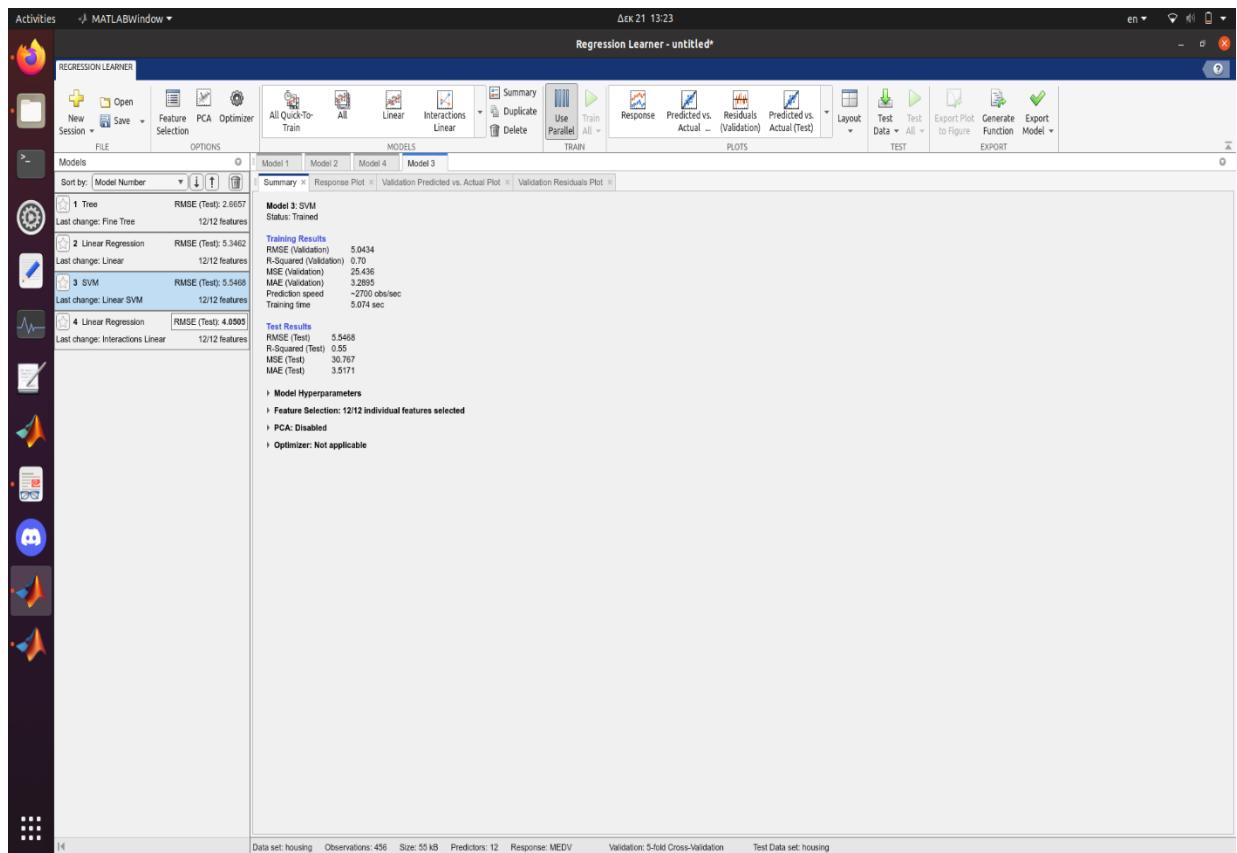


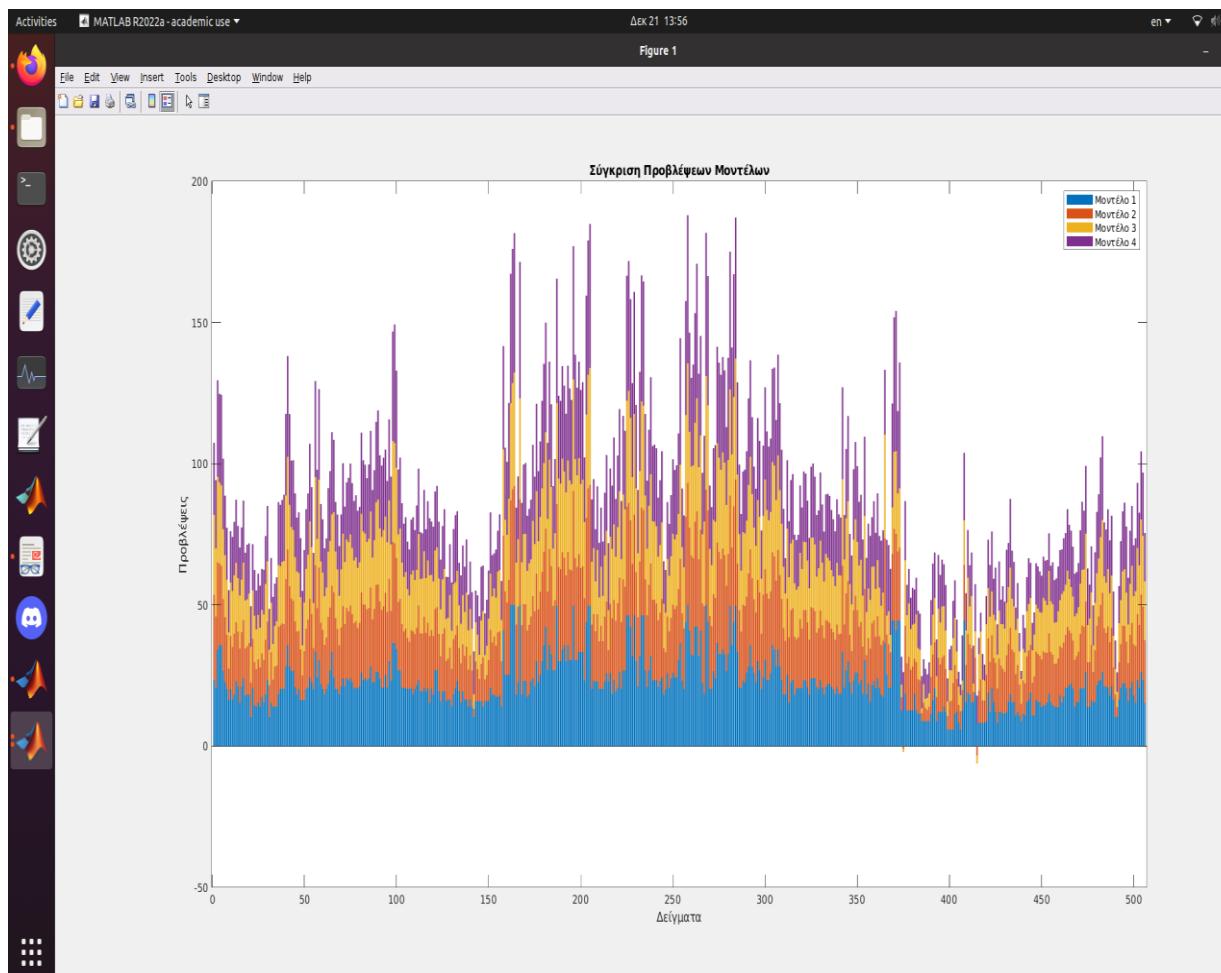
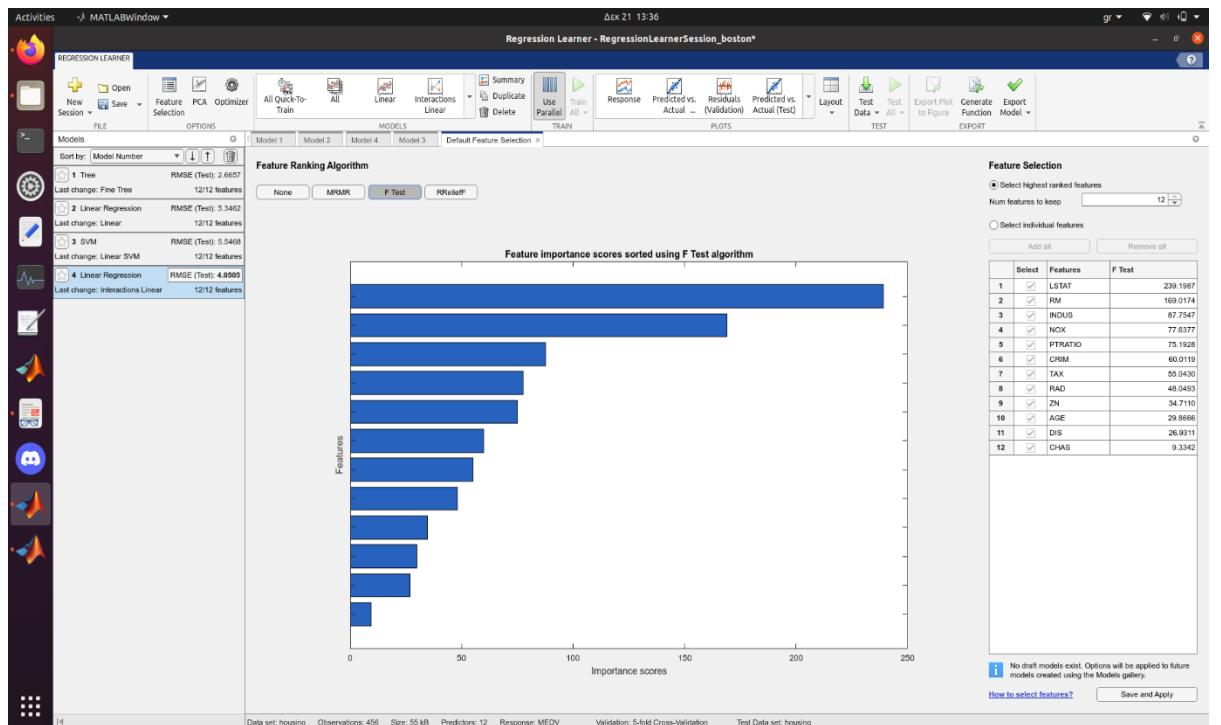












Μέρος Β

Το Β μέρος της εργασίας ασχολείται με δεδομένα που προκύπτουν από παρατηρήσεις ανθρώπινης δραστηριότητας με τη βοήθεια αισθητήρα. Ειδικότερα, όπως είδαμε και στο εργαστήριο-παρακολούθηση βίντεο του πειράματος, λαμβάνονται ορισμένες πληροφορίες μέσω κινητού τηλεφώνου που αφορούν δραστηριότητες του κατόχου του κινητού και είναι οι εξής: Laying, Sitting, Climbing Stairs, Standing, Walking.

Μεγάλης σημασίας δεδομένα για την άσκηση είναι τα εξής: rawSensorData_train.mat και rawSensorData_test.mat. Αρχικά, κατεβάζουμε μέσω eclass το dataset UCI HAR.

Το συγκεκριμένο παράδειγμα ανήκει στην κατηγορία προβλημάτων ταξινόμησης, οπότε χρειάζεται χρήση του Classification Learner. Στόχος του παραδείγματος είναι να παρέχει βήματα για τη δημιουργία ενός ταξινομητή, που προσδιορίζει αυτόματα την εκάστοτε ανθρώπινη δραστηριότητα με τη βοήθεια των μετρήσεων του αισθητήρα.

Πρέπει να σημειωθεί ότι γίνεται χρήση ορισμένων έτοιμων συναρτήσεων που υπάρχουν στο dataset: plotRawSensorData, plotActivityResults, Wmean, Wpca1, Wstd.

Ανάλυση Αποτελεσμάτων

Μέσω του Classification Learner εισαγάγαμε τα δεδομένα που χρειαζόταν(predictors-response), επιλέξαμε 20% Holdout Validation και στη συνέχεια επιλέξαμε 3 μοντέλα: Fine Tree, Fine KNN(5 γείτονες) και Linear SVM. Το αποτελεσματικότερο μοντέλο ήταν το Fine KNN με Accuracy 94.4%. Για το συγκεκριμένο μοντέλο σημειώσαμε τις περισσότερες παρατηρήσεις για την κατάσταση Climbing Stairs(371).

Επίσης, παρατηρούμε ότι η απόδοση του μοντέλου για κάθε μία από τις 5 καταστάσεις που θα βρεθεί ο άνθρωπος είναι: Laying 100%, Sitting 95.3%, Climbing Stairs 90%, Standing 95.6%, Walking 92.7%. Επιπροσθέτως, από τον πίνακα σύγχυσης παρατηρούμε ότι το μεγαλύτερο ποσοστό των λανθασμένων ταξινομήσεων οφείλεται στο μπέρδεμα των καταστάσεων Climbing Stairs και Walking.

Σημαντικές συναρτήσεις είναι οι εξής:

- plotRawSensorData: Ως είσοδο δέχεται τα predictors και παρουσιάζει τα δεδομένα-καταστάσεις που βρίσκεται ο άνθρωπος την εκάστοτε στιγμή, όπως λαμβάνονται από τον αισθητήρα.
- plotActivityResults: Στο ίδιο διάγραμμα παρουσιάζονται οι προβλεπόμενες, αλλά και οι πραγματικές καταστάσεις που βρίσκεται ο άνθρωπος που διεξάγει το πείραμα σε κάθε στιγμή. Εάν συμφωνούν εμφανίζονται και οι 2 με πράσινο. Ενώ, όταν διαφωνούν το κείμενο της πρόβλεψης εμφανίζεται με κόκκινο χρώμα.

Ακολουθούν ο κώδικας και σχετικά screenshots.

Kώδικας:

B_final.m:

```
if ~exist('UCI HAR Dataset','file')
    downloadSensorData;
end

if
~exist(['/home/dgour/Desktop/PatReg_MachineLearning/erg3/B_final/HumanActivity/',...
        'UCI HAR Dataset/rawSensorData_train.mat'],'file') &&
~exist(['/home/dgour/Desktop/',...
        'PatReg_MachineLearning/erg3/B_final/HumanActivity/UCI HAR Dataset',...
        '/rawSensorData_test.mat'],'file')
    saveSensorDataAsMATFiles;
end

load rawSensorData_train

plotRawSensorData(total_acc_x_train, total_acc_y_train, ...
    total_acc_z_train, trainActivity, 1000)

rawSensorDataTrain = table',...
    total_acc_x_train, total_acc_y_train, total_acc_z_train, ...
    body_gyro_x_train, body_gyro_y_train, body_gyro_z_train);

humanActivityData = varfun(@Wmean, rawSensorDataTrain);
humanActivityData.activity = trainActivity;

% pre-process
T_mean = varfun(@Wmean, rawSensorDataTrain);
T_stdv = varfun(@Wstd, rawSensorDataTrain);
T_pca = varfun(@Wpcal, rawSensorDataTrain);

humanActivityData = [T_mean, T_stdv, T_pca];
humanActivityData.activity = trainActivity;

%%%%% Χρήση Classification Learner %%%%

% -----
load rawSensorData_test

rawSensorDataTest = table',...
    total_acc_x_test, total_acc_y_test, total_acc_z_test, ...
    body_gyro_x_test, body_gyro_y_test, body_gyro_z_test);

T_mean = varfun(@Wmean, rawSensorDataTest);
T_stdv = varfun(@Wstd, rawSensorDataTest);
T_pca = varfun(@Wpcal, rawSensorDataTest);

humanActivityData = [T_mean, T_stdv, T_pca];
humanActivityData.activity = testActivity;

%%%%% Χρήση Classification Learner %%%%
```

```

load('/home/dgour/Desktop/PatReg_MachineLearning/erg3/B_final/HumanActivity/trainedModel2.mat')
load('/home/dgour/Desktop/PatReg_MachineLearning/erg3/B_final/HumanActivity/trainedModel1.mat')
load('/home/dgour/Desktop/PatReg_MachineLearning/erg3/B_final/HumanActivity/trainedModel3.mat')

plotActivityResults(trainClassifier2, rawSensorDataTest, humanActivityData, 0.1)
% Code generated by Classification Learner
% trainClassifier1:Fine Tree
% trainClassifier2:Fine KNN
% trainClassifier3:Linear SVM

```

downloadSensorData.m

```

function downloadSensorData
% Download and extract data if data folder does not exist
% Copyright (c) 2015, MathWorks, Inc.

if ~exist('UCI HAR Dataset','file') == 7
    downloadlink = 'https://archive.ics.uci.edu/ml/machine-learning-databases/00240/UCI%20HAR%20Dataset.zip';
    tic
    disp('UCI HAR Dataset does not exist, downloading dataset... ')
    fname = websave('UCI_HAR_Dataset.zip',downloadlink);
    toc
    disp('Done downloading, file downloaded to:')
    disp(fname)
    disp(' ')
    disp('It may be faster to extract the zip file manually using another software,')
    yn = input('Do you want MATLAB to extract the file for you (y/n)? ','s');
    if strcmp(yn,'y')
        tic
        disp('Extracting file, this may take a while... ')
        foldername = unzip(fname);
        disp('Done extracting')
        toc
    else
        disp(' ')
        disp('OK, You must manually extract the file contents to the current folder before proceeding.')
        disp('Please make sure that you don''t change the default folder name: ''UCI HAR Dataset'''')
    end
end

```

saveSensorDataAsMATFiles.m

```
function saveSensorDataAsMATFiles
% Load raw sensor data from files and convert them into tables
% The function saves the raw sensor data into two mat files:
% # rawSensorData_train
% # rawSensorData_test
% Copyright (c) 2015, MathWorks, Inc.

if exist('rawSensorData_train.mat','file') &&
exist('rawSensorData_test.mat','file')
    fprintf(1,'rawSensorData_train.mat and rawSensorData_test.mat
already exists at location:\n');
    disp(['*' , which('rawSensorData_train.mat')]);
    disp(['*' , which('rawSensorData_test.mat')]);
    disp(' ')
else
    %% Load training data from files
    activity_labels =
{'Walking','WalkingUpstairs','WalkingDownstairs','Sitting','Standing',
'Laying'};
    trainActivity = categorical(importdata('UCI HAR
Dataset\train\y_train.txt'),1:6,activity_labels);
    trainActivity =
mergecats(trainActivity,{ 'WalkingUpstairs','WalkingDownstairs'},'Clim
bingStairs');
    trainActivity = reordercats(trainActivity
,{'Laying','Sitting','ClimbingStairs','Standing','Walking'});
    % Choose this if you want to only load the total acca and gyro
data
    filestoload = strcat('UCI HAR Dataset\train\Inertial
Signals\',{ '*gyro*', 'total*' });
    % Choose this if you want to load all files
    % filestoload = strcat('UCI HAR Dataset\train\Inertial
Signals\' );
    disp('Loading training data from files:')
    try
        dstrain =
datastore(filestoload,'TextscanFormats', repmat({'%f'},1,128),
'ReadVariableNames',false);
        catch err
            if
strcmp(err.identifier,'MATLAB: datastoreio: pathlookup: fileNotFound')
                error('File not found. Please make sure that you download
and extract the data first using ''downloadSensorData'' function')
            end
        end
        dstrain.ReadSize = 'file';
        [~,fnames] =
cellfun(@fileparts,dstrain.Files,'UniformOutput',false);
        iter = 1;
        while hasdata(dstrain)
            fprintf('Importing: %16s ...',fnames{iter})
            M = table2array(read(dstrain));
            rawSensorDataTrain.(fnames{iter}) = M;
            iter = iter + 1;
            fprintf('Done\n')
        end
    end
```

```

rawSensorDataTrain.trainActivity = trainActivity;
disp(' ')
%% Load test data from files
testActivity = categorical(importdata('UCI HAR
Dataset\test\y_test.txt'),1:6,activity_labels);
testActivity =
mergecats(testActivity,['WalkingUpstairs','WalkingDownstairs'],'Climb
ingStairs');
testActivity = reordercats(testActivity
,{'Laying','Sitting','ClimbingStairs','Standing','Walking'});
% Choose this if you want to only load the total acc and gyro data
filestoload = strcat('UCI HAR Dataset\test\Inertial
Signals\',{'*gyro*', 'total*'});
% Choose this if you want to load all files
% filestoload = strcat('UCI HAR Dataset\test\Inertial Signals\'');
disp('Loading test data from files:')
dstest =
datastore(filestoload,'TextscanFormats', repmat({'%f'},1,128), 'Datasto
reType','tabulartext',...
'ReadVariableNames',false);
[~,fnames] =
cellfun(@fileparts,dstest.Files,'UniformOutput',false);
dstest.ReadSize = 'file';
iter = 1;
while hasdata(dstest)
    fprintf('Importing: %16s ...',fnames{iter})
    M = table2array(read(dstest));
    rawSensorDataTest.(fnames{iter}) = M;
    iter = iter + 1;
    fprintf('Done\n')
end
rawSensorDataTest.testActivity = testActivity;
disp(' ')
% Saving MAT file with raw data
fprintf('Saving MAT files: rawSensorData_train.mat ...')
save rawSensorData_train.mat -struct rawSensorDataTrain
disp('Done')
fprintf('Saving MAT files: rawSensorData_test.mat ...')
save rawSensorData_test.mat -struct rawSensorDataTest
disp('Done')
end

```

Wmean.m

```

function Y = Wmean(X)
% Copyright (c) 2015, MathWorks, Inc.
Y = mean(X,2);
end

```

Wpcal.m

```

function Y = Wpcal(X)
% Copyright (c) 2015, MathWorks, Inc.
[~,Y] = pca(X, 'NumComponents', 1);
end

```

Wstd.m

```
function Y = Wstd(X)
% Copyright (c) 2015, MathWorks, Inc.
Y = std(X, [], 2);
end
```

HumanActivityLearning.m

```
%% Human Activity Learning Using Mobile Phone Data
% Human activity sensor data contains observations derived from
% sensor measurements taken from smartphones worn by people while
doing
% different activities (walking, lying, sitting etc). The goal of
this
% example is to provide a strategy to build a classifier that can
% automatically identify the activity type given the sensor
measurements.
%
% Copyright (c) 2015, MathWorks, Inc.

%% Description of the Data
% The dataset consists of accelerometer and gyroscope data captured
at
% 50Hz. The raw sensor data contain fixed-width sliding windows of
2.56 sec
% (128 readings/window). The activities performed by the subject
include:
% 'Walking', 'ClimbingStairs', 'Sitting', 'Standing', and 'Laying'
%%
% *How to get the data:*
% Execute |downloadSensorData| and follow the instructions to
download the
% and extract the data from the source webpage. After the files have
been
% extracted run |saveSensorDataAsMATFiles|. This will create two MAT
files:
% |rawSensorData_train| and |rawSensorData_test| with the raw sensor
data
%%
% # *total_acc_(x/y/z)_train :* Raw accelerometer sensor data
% # *body_gyro_(x/y/z)_train :* Raw gyroscope sensor data
% # *trainActivity :* Training data labels
% # *testActivity :* Test data labels

%%
% Reference:
%
% |Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra and
Jorge L.
% Reyes-Ortiz. Human Activity Recognition on Smartphones using a
% Multiclass Hardware-Friendly Support Vector Machine. International
% Workshop of Ambient Assisted Living (IWAAL 2012). Vitoria-Gasteiz,
% Spain. Dec 2012|
```

```

% execute these functions.
%%
% * |downloadSensorData| : This function will download the dataset
and
% extract its contents to a folder called: UCI HAR Dataset
% This folder must be present before you execute
|saveSensorDataAsMATFiles|
if ~exist('UCI HAR Dataset','file')
    downloadSensorData;
end

%% Load data frome individual files and save as MAT file for reuse
%%
% * |saveSensorDataAsMATFiles| : This function will load the data
from the individual
% source files and save the data in a single MAT file for easy
accesss
if ~exist('rawSensorData_train.mat','file') &&
~exist('rawSensorData_test.mat','file')
    saveSensorDataAsMATFiles;
end

%% Load Training Data
load rawSensorData_train

%% Display data summary
plotRawSensorData(total_acc_x_train, total_acc_y_train, ...
    total_acc_z_train,trainActivity,1000)

%% Create Table variable
rawSensorDataTrain = table',...
    total_acc_x_train, total_acc_y_train, total_acc_z_train, ...
    body_gyro_x_train, body_gyro_y_train, body_gyro_z_train);

%% Pre-process Training Data: *Feature Extraction*
% Lets start with a simple preprocessing technique. Since the raw
sensor
% data contain fixed-width sliding windows of 2.56sec (128
readings/window)
% lets start with a simple average feature for every 128 points

humanActivityData = varfun(@Wmean,rawSensorDataTrain);
humanActivityData.activity = trainActivity;

%% Train a model and assess its performance using Classification
Learner
classificationLearner

%% Additional Feature Extraction

T_mean = varfun(@Wmean, rawSensorDataTrain);
T_stdv = varfun(@Wstd,rawSensorDataTrain);
T_pca = varfun(@Wpcal,rawSensorDataTrain);

humanActivityData = [T_mean, T_stdv, T_pca];
humanActivityData.activity = trainActivity;

%% Use the new features to train a model and assess its performance
classificationLearner
%%
%
```

```

% <<classificationLearner.png>>
%
%% Load Test Data
load rawSensorData_test

%% Visualize classifier performance on test data
%
% Step 1: Create a table
rawSensorDataTest = table(...  

    total_acc_x_test, total_acc_y_test, total_acc_z_test, ...  

    body_gyro_x_test, body_gyro_y_test, body_gyro_z_test);

% Step 2: Extract features from raw sensor data
T_mean = varfun(@Wmean, rawSensorDataTest);  

T_stdv = varfun(@Wstd, rawSensorDataTest);  

T_pca = varfun(@Wpca1, rawSensorDataTest);

humanActivityData = [T_mean, T_stdv, T_pca];  

humanActivityData.activity = testActivity;

% Step 3: Use trained model to predict activity on new sensor data
% Make sure that you've exported 'trainedClassifier' from
% ClassificationLearner
plotActivityResults(trainedClassifier, rawSensorDataTest, humanActivity  

Data, 0.1)

%%
%
% <<PredictionResults.png>>
%

```

plotActivityResults.m

```

function
plotActivityResults(mdl, rawSensorDataTest, humanActivityTest, delay)
% Use trained model to predict activity on new sensor data
% Copyright (c) 2015, MathWorks, Inc.
if nargin < 4
    delay = 0.02;
end
g = 9.81; % in m/s^2
time = linspace(0, 2.56, 128);

fig = figure('Name', 'Human Activity  
Detection', 'NumberTitle', 'off', 'Visible', 'off');
fig.Position(3:4) = [600, 400];
movegui('center')
fig.Visible = 'on';

ax1 = subplot(2, 1, 1, 'Parent', fig, 'Xgrid', 'on', 'Ygrid', 'on', ...
    'XLim', [time(1) time(end)], 'YLim', [-2*g 2*g]);
ax2 = subplot(2, 1, 2, 'Parent', fig, 'Xgrid', 'on', 'Ygrid', 'on', ...
    'XLim', [time(1) time(end)], 'YLim', [-2 2]);
% axis(ax1, 'square'), axis(ax2, 'square')

clr = get(gcf, 'DefaultAxesColorOrder');
L(1) =
line(time, g*rawSensorDataTest.total_acc_x_test(1, :), 'color', clr(1, :),

```

```

'Parent',ax1,'LineWidth',1.5,'DisplayName','Accelerometer X');
L(2) =
line(time,g*rawSensorDataTest.total_acc_y_test(1,:), 'color',clr(2,:),
'Parent',ax1,'LineWidth',1.5,'DisplayName','Accelerometer Y');
L(3) =
line(time,g*rawSensorDataTest.total_acc_z_test(1,:), 'color',clr(3,:),
'Parent',ax1,'LineWidth',1.5,'DisplayName','Accelerometer Z');

L(4) = line(time,
rawSensorDataTest.body_gyro_x_test(1,:), 'color',clr(4,:),'Parent',ax2
,'LineWidth',1.5,'DisplayName','Gyroscope X');
L(5) = line(time,
rawSensorDataTest.body_gyro_y_test(1,:), 'color',clr(5,:),'Parent',ax2
,'LineWidth',1.5,'DisplayName','Gyroscope Y');
L(6) = line(time,
rawSensorDataTest.body_gyro_z_test(1,:), 'color',clr(6,:),'Parent',ax2
,'LineWidth',1.5,'DisplayName','Gyroscope Z');

xlabel(ax1,'Time (s)')
ylabel(ax1,'(Accelerometer Readings (m \cdot s^{-2}))')
legend(ax1,'show')
title(ax1,sprintf('Human Activity Mobile Sensor Data'));

xlabel(ax2,'Time (s)')
ylabel(ax2,'Gyroscope Readings rad \cdot sec{-1}')
legend(ax2,'show')
title(ax2,['Classifier: ', getClassifierName(mdl)]);

ann1 = annotation(fig,'textbox',[ax1.Position(1:3) 0.04],...
    'String','Predicted Activity :'
    'NA','FontSize',12,'FitBoxToText','off',...
    'BackgroundColor',[0 0.7
0.3],'HorizontalAlignment','Center','VerticalAlignment','middle','FaceAlpha',0.5);
ann2 = annotation(fig,'textbox',[ax1.Position(1) ax1.Position(2)+0.04
ax1.Position(3) 0.04],...
    'String','Actual Activity :'
    'NA','FontSize',12,'FitBoxToText','off',...
    'BackgroundColor',[0 0.7
0.3],'HorizontalAlignment','Center','VerticalAlignment','middle','FaceAlpha',0.5);

%% Loop through the raw data and plot the sensor values
try
for ii = 600:height(humanActivityTest)

    activity = predict(mdl,humanActivityTest{ii,1:end-1});

    if activity == humanActivityTest.activity(ii)
        predclr = [0 0.7 0.3];
    else
        predclr = [1 0 0];
    end
    set(ann1,'String',['Predicted Activity : ' char(activity)],...
        'BackgroundColor',predclr);
    set(ann2,'String',['Actual Activity : '
char(humanActivityTest.activity(ii))],...
        'BackgroundColor',[0 0.7 0.3]);

    L(1).YData = g*rawSensorDataTest.total_acc_x_test(ii,:);
    L(2).YData = g*rawSensorDataTest.total_acc_y_test(ii,:);

```

```

L(3).YData = g*rawSensorDataTest.total_acc_z_test(ii,:);

L(4).YData = rawSensorDataTest.body_gyro_x_test(ii,:);
L(5).YData = rawSensorDataTest.body_gyro_y_test(ii,:);
L(6).YData = rawSensorDataTest.body_gyro_z_test(ii,:);

drawnow
pause(delay)
end
catch err
end

function cname = getClassifierName(trainedClassifier)
    cname = class(trainedClassifier);
    if isa(trainedClassifier,'ClassificationECOC')
        cname = 'SVM';
    end
    pos = strfind(cname,'.');
    if ~isempty(pos)
        cname = cname(pos(end)+1:end);
    end
end

```

plotModelResults.m

```

function plotModelResults(mdl,humanActivityTest, delay)
% Use trained model to predict activity on new sensor data
% Copyright (c) 2015, MathWorks, Inc.

load('rawSensorData_test')
rawSensorDataTest = table...
    total_acc_x_test, total_acc_y_test, total_acc_z_test, ...
    body_gyro_x_test, body_gyro_y_test, body_gyro_z_test);

g = 9.81; % in m/s^2
time = linspace(0,2.56,128);

fig = figure('Name','Human Activity
Detection','NumberTitle','off','Visible','off');
fig.Position(3:4) = 600;
movegui('center')
fig.Visible = 'on';

ax1 = subplot(2,1,1,'Parent',fig,'Xgrid','on','Ygrid','on',...
    'XLim',[time(1) time(end)],'YLim',[-2*g 2*g]);
ax2 = subplot(2,1,2,'Parent',fig,'Xgrid','on','Ygrid','on',...
    'XLim',[time(1) time(end)],'YLim',[-2 2]);
% axis(ax1,'square'), axis(ax2,'square')

clr = get(groot,'DefaultAxesColorOrder');
L(1) =
line(time,g*rawSensorDataTest.total_acc_x_test(1,:), 'color',clr(1,:),
'Parent',ax1,'LineWidth',1.5,'DisplayName','Accelerometer X');
L(2) =
line(time,g*rawSensorDataTest.total_acc_y_test(1,:), 'color',clr(2,:),
'Parent',ax1,'LineWidth',1.5,'DisplayName','Accelerometer Y');
L(3) =
line(time,g*rawSensorDataTest.total_acc_z_test(1,:), 'color',clr(3,:),
'Parent',ax1,'LineWidth',1.5,'DisplayName','Accelerometer Z');

```

```

L(4) = line(time,
rawSensorDataTest.body_gyro_x_test(1,:), 'color',clr(4,:),'Parent',ax2
,'LineWidth',1.5,'DisplayName','Gyroscope X');
L(5) = line(time,
rawSensorDataTest.body_gyro_y_test(1,:), 'color',clr(5,:),'Parent',ax2
,'LineWidth',1.5,'DisplayName','Gyroscope Y');
L(6) = line(time,
rawSensorDataTest.body_gyro_z_test(1,:), 'color',clr(6,:),'Parent',ax2
,'LineWidth',1.5,'DisplayName','Gyroscope Z');

xlabel(ax1,'Time (s)')
ylabel(ax1,'Accelerometer Readings (m \cdot s^{-2})')
legend(ax1,'show')
title(ax1,sprintf('Human Activity Mobile Sensor Data'));

xlabel(ax2,'Time (s)')
ylabel(ax2,'Gyroscope Readings rad \cdot sec^{-1}')
legend(ax2,'show')
title(ax2,['Classifier: ', getClassifierName(mdl)]);

ann1 = annotation(fig,'textbox',[ax1.Position(1:3) 0.04],...
    'String','Predicted Activity : ...
NA','FontSize',12,'FitBoxToText','off',...
    'BackgroundColor',[0 0.7 ...
0.3],'HorizontalAlignment','Center','VerticalAlignment','middle','FaceAlpha',0.5);
ann2 = annotation(fig,'textbox',[ax1.Position(1) ax1.Position(2)+0.04 ...
ax1.Position(3) 0.04],...
    'String','Actual Activity : ...
NA','FontSize',12,'FitBoxToText','off',...
    'BackgroundColor',[0 0.7 ...
0.3],'HorizontalAlignment','Center','VerticalAlignment','middle','FaceAlpha',0.5);

%% Loop through the raw data and plot the sensor values
try
for ii = 600:height(humanActivityTest)

    activity = predict(mdl,humanActivityTest{ii,1:end-1});

    if activity == humanActivityTest.activity(ii)
        predclr = [0 0.7 0.3];
    else
        predclr = [1 0 0];
    end
    set(ann1,'String',['Predicted Activity : ' char(activity)],...
        'BackgroundColor',predclr);
    set(ann2,['Actual Activity : ' ...
char(humanActivityTest.activity(ii))],...
        'BackgroundColor',[0 0.7 0.3]);

    L(1).YData = g*rawSensorDataTest.total_acc_x_test(ii,:);
    L(2).YData = g*rawSensorDataTest.total_acc_y_test(ii,:);
    L(3).YData = g*rawSensorDataTest.total_acc_z_test(ii__);

    L(4).YData = rawSensorDataTest.body_gyro_x_test(ii,:);
    L(5).YData = rawSensorDataTest.body_gyro_y_test(ii,:);
    L(6).YData = rawSensorDataTest.body_gyro_z_test(ii__);

drawnow

```

```

        pause(delay)
end
catch err %#ok<NASGU>
end

function cname = getClassifierName(trainedClassifier)
    cname = class(trainedClassifier);
    if isa(trainedClassifier,'ClassificationECOC')
        cname = 'SVM';
    end
    pos = strfind(cname,'.');
    if ~isempty(pos)
        cname = cname(pos(end)+1:end);
    end

```

plotRawSensorData.m

```

function
plotRawSensorData(rawSensor1,rawSensor2,rawSensor3,activity,n_obs)

if nargin < 5
    n_obs = size(rawSensor1,1);
end
g = 9.81;

rawTS(:,1) = reshape(rawSensor1(1:n_obs,:)',[],1);
rawTS(:,2) = reshape(rawSensor2(1:n_obs,:)',[],1);
rawTS(:,3) = reshape(rawSensor3(1:n_obs,:)',[],1);

activityType =
repelem(activity(1:n_obs),128*ones(numel(activity(n_obs)),1));

fig = figure('Name','Human Activity
Detection','NumberTitle','off',...
    'Units','Normalized','Visible','off');
fig.Position(3:4) = [0.7,0.95];
movegui('center')
fig.Visible = 'on';

ax(1) = subplot(3,1,1,'Parent',fig,'Xgrid','on','Ygrid','on',...
    'YLim',[-3*g 2*g]);
ax(2) = subplot(3,1,2,'Parent',fig,'Xgrid','on','Ygrid','on',...
    'YLim',[-3*g 2*g]);
ax(3) = subplot(3,1,3,'Parent',fig,'Xgrid','on','Ygrid','on',...
    'YLim',[-3*g 2*g]);

clr = get(groot,'DefaultAxesColorOrder');
axes(ax(1)),
gscatter(1:size(rawTS,1),g*rawTS(:,1),activityType,clr,'.',8,'on')
axes(ax(2)),
```

```

gscatter(1:size(rawTS,1),g*rawTS(:,2),activityType,clr,'.',8,'off')
axes(ax(3)),
gscatter(1:size(rawTS,1),g*rawTS(:,3),activityType,clr,'.',8,'off')

title(ax(1),inputname(1),'Interpreter','none')
title(ax(2),inputname(2),'Interpreter','none')
title(ax(3),inputname(3),'Interpreter','none')

linkaxes(ax,'x')

grid(ax(1),'on')
grid(ax(2),'on')
grid(ax(3),'on')

ylabel(ax(1),'(Acc-X (m \cdot s^{-2}))')
ylabel(ax(2),'(Acc-Y (m \cdot s^{-2}))')
ylabel(ax(3),'(Acc-Z (m \cdot s^{-2}))')

xlabel(ax(1),'Time (sec)'), ax(1).XTickLabel = [];
xlabel(ax(2),'Time (sec)'), ax(2).XTickLabel = [];
xlabel(ax(3),'Time (sec)'), ax(3).XTickLabel = [];

end

```

trainClassifier2.m

```

function [trainedClassifier, validationAccuracy] =
trainClassifier2(trainingData)
% [trainedClassifier, validationAccuracy] =
trainClassifier(trainingData)
% Returns a trained classifier and its accuracy. This code recreates
the
% classification model trained in Classification Learner app. Use the
% generated code to automate training the same model with new data,
or to
% learn how to programmatically train models.
%
% Input:
%   trainingData: A table containing the same predictor and
response
%       columns as those imported into the app.
%
% Output:
%   trainedClassifier: A struct containing the trained classifier.
The
%       struct contains various fields with information about the
trained
%       classifier.
%
%   trainedClassifier.predictFcn: A function to make predictions
on new
%       data.

```

```

%
% validationAccuracy: A double containing the accuracy as a
% percentage. In the app, the Models pane displays this overall
% accuracy score for each model.
%
% Use the code to train the model with new data. To retrain your
% classifier, call the function from the command line with your
original
% data or new data as the input argument trainingData.
%
% For example, to retrain a classifier trained with the original data
set
% T, enter:
% [trainedClassifier, validationAccuracy] = trainClassifier(T)
%
% To make predictions with the returned 'trainedClassifier' on new
data T2,
% use
% yfit = trainedClassifier.predictFcn(T2)
%
% T2 must be a table containing at least the same predictor columns
as used
% during training. For details, enter:
% trainedClassifier.HowToPredict

% Auto-generated by MATLAB on 26-Dec-2023 16:50:42

%
% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'Wmean_total_acc_x_train',
'Wmean_total_acc_y_train', 'Wmean_total_acc_z_train',
'Wmean_body_gyro_x_train', 'Wmean_body_gyro_y_train',
'Wmean_body_gyro_z_train', 'Wstd_total_acc_x_train',
'Wstd_total_acc_y_train', 'Wstd_total_acc_z_train',
'Wstd_body_gyro_x_train', 'Wstd_body_gyro_y_train',
'Wstd_body_gyro_z_train', 'Wpcal_total_acc_x_train',
'Wpcal_total_acc_y_train', 'Wpcal_total_acc_z_train',
'Wpcal_body_gyro_x_train', 'Wpcal_body_gyro_y_train',
'Wpcal_body_gyro_z_train'};
predictors = inputTable(:, predictorNames);
response = inputTable.activity;
isCategoricalPredictor = [false, false, false, false, false, false,
false, false, false, false, false, false, false, false, false,
false, false];

%
% Train a classifier
% This code specifies all the classifier options and trains the
classifier.
classificationKNN = fitcknn(...  

    predictors, ...
    response, ...
    'Distance', 'Euclidean', ...
    'Exponent', [], ...
    'NumNeighbors', 5, ...
    'DistanceWeight', 'Equal', ...
    'Standardize', true, ...
    'ClassNames', categorical({'Laying'; 'Sitting'; 'ClimbingStairs';
'Standing'; 'Walking'}, {'Laying' 'Sitting' 'ClimbingStairs'

```

```

'Standing' 'Walking')));

% Create the result struct with predict function
predictorExtractionFcn = @(t) t(:, predictorNames);
knnPredictFcn = @(x) predict(classificationKNN, x);
trainedClassifier.predictFcn = @(x)
knnPredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedClassifier.RequiredVariables = {'Wmean_body_gyro_x_train',
'Wmean_body_gyro_y_train', 'Wmean_body_gyro_z_train',
'Wmean_total_acc_x_train', 'Wmean_total_acc_y_train',
'Wmean_total_acc_z_train', 'Wpcal_body_gyro_x_train',
'Wpcal_body_gyro_y_train', 'Wpcal_body_gyro_z_train',
'Wpcal_total_acc_x_train', 'Wpcal_total_acc_y_train',
'Wpcal_total_acc_z_train', 'Wstd_body_gyro_x_train',
'Wstd_body_gyro_y_train', 'Wstd_body_gyro_z_train',
'Wstd_total_acc_x_train', 'Wstd_total_acc_y_train',
'Wstd_total_acc_z_train'};
trainedClassifier.ClassificationKNN = classificationKNN;
trainedClassifier.About = 'This struct is a trained model exported
from Classification Learner R2022a.';
trainedClassifier.HowToPredict = sprintf('To make predictions on a
new table, T, use: \n yfit = c.predictFcn(T) \nreplacing ''c'' with
the name of the variable that is this struct, e.g. ''trainedModel''.
\n \nThe table, T, must contain the variables returned by: \n
c.RequiredVariables \nVariable formats (e.g. matrix/vector, datatype)
must match the original training data. \nAdditional variables are
ignored. \n \nFor more information, see <a
href="matlab:helpview(fullfile(docroot, ''stats'', ''stats.map''),
''appclassification_exportmodeltoworkspace'')">How to predict using
an exported model</a>');
% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'Wmean_total_acc_x_train',
'Wmean_total_acc_y_train', 'Wmean_total_acc_z_train',
'Wmean_body_gyro_x_train', 'Wmean_body_gyro_y_train',
'Wmean_body_gyro_z_train', 'Wstd_total_acc_x_train',
'Wstd_total_acc_y_train', 'Wstd_total_acc_z_train',
'Wstd_body_gyro_x_train', 'Wstd_body_gyro_y_train',
'Wstd_body_gyro_z_train', 'Wpcal_total_acc_x_train',
'Wpcal_total_acc_y_train', 'Wpcal_total_acc_z_train',
'Wpcal_body_gyro_x_train', 'Wpcal_body_gyro_y_train',
'Wpcal_body_gyro_z_train'};
predictors = inputTable(:, predictorNames);
response = inputTable.activity;
isCategoricalPredictor = [false, false, false, false, false, false,
false, false, false, false, false, false, false, false, false,
false, false];
% Set up holdout validation
cvp = cvpartition(response, 'Holdout', 0.2);
trainingPredictors = predictors(cvp.training, :);
trainingResponse = response(cvp.training, :);
trainingIsCategoricalPredictor = isCategoricalPredictor;

% Train a classifier
% This code specifies all the classifier options and trains the

```

```

classifier.
classificationKNN = fitcknn(...
    trainingPredictors, ...
    trainingResponse, ...
    'Distance', 'Euclidean', ...
    'Exponent', [], ...
    'NumNeighbors', 5, ...
    'DistanceWeight', 'Equal', ...
    'Standardize', true, ...
    'ClassNames', categorical({'Laying'; 'Sitting'; 'ClimbingStairs';
'Standing'; 'Walking'}), {'Laying' 'Sitting' 'ClimbingStairs'
'Standing' 'Walking'}));

% Create the result struct with predict function
knnPredictFcn = @(x) predict(classificationKNN, x);
validationPredictFcn = @(x) knnPredictFcn(x);

% Add additional fields to the result struct

% Compute validation predictions
validationPredictors = predictors(cvp.test, :);
validationResponse = response(cvp.test, :);
[validationPredictions, validationScores] =
validationPredictFcn(validationPredictors);

% Compute validation accuracy
correctPredictions = (validationPredictions == validationResponse);
isMissing = ismissing(validationResponse);
correctPredictions = correctPredictions(~isMissing);
validationAccuracy =
sum(correctPredictions)/length(correctPredictions);

```

Screenshots:

