Queen Mary University of London

Department of Electrical Engineering & Computer Science

ECS797P Machine Learning for Visual Data Analysis

Lab 3 – Age Estimation by Regression

Hasan Emre Erdemoglu

1 April 2021

Getting Started

This assignment uses FG-NET Aging Dataset. The dataset consists of Active Appearance Model (AAM) features of 1002 face images, 500 reserved for training and 502 reserved for testing purposes. AAM features have 201 dimensions.

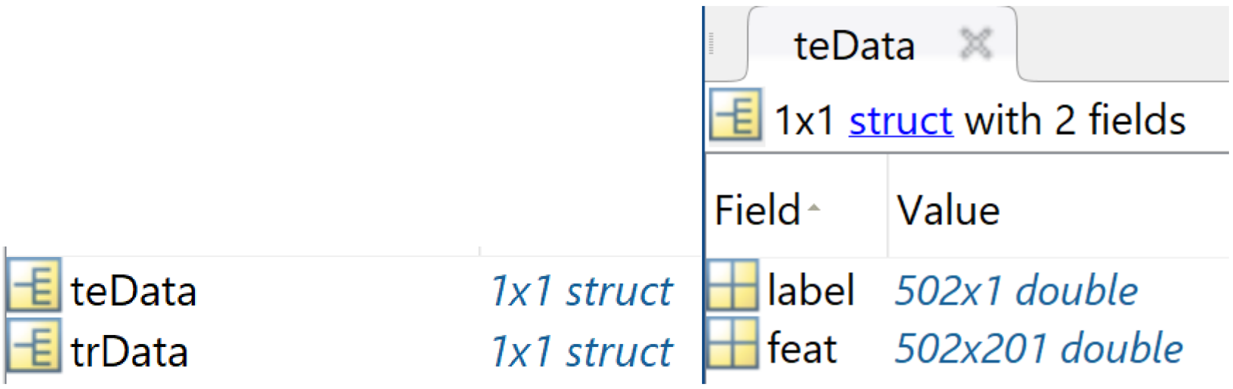
The dataset contains images of 82 people from different ethnicities. Each people have approximately 12 photos with different lighting, pose and expressions. The dataset is labelled with respect to age of the people ranging from 0 to 69, ordered chronologically[1].

Lab 3: Age Estimation by Regression:

Parts 1,2 and 3 is already implemented by the assignment and therefore the implementation details will be described briefly. The entire code listing can be found in the appendix. Codes for each Question are also separated with comments within the listing. The necessary code segments are listed within the report if further clarification is needed.

Question 1:

The AAM feature vector and label pair is given by the file *data_age.mat*. This file could be read by the MATLAB function *load(...)*. This command returns two struct variables testing and training data; *teData* and *trData*, respectively. These struct variables have two variables named *label* (that is a vector with length *N)* and *feat* (that is a matrix of size N by 201), *N* being the number of samples in the training and testing data.



teData	1x1 struct
trData	1x1 struct

teData

1x1 struct with 2 fields

Field	Value
label	502x1 double
feat	502x201 double

Figure 1. Struct variables and their inner structure (Example on teData)

The code for this section can be found in the listing.

[1] The dataset information extracted from Lab 3 document.

Question 2:

This section uses MATLAB function *regress(…)* to construct a linear regression model. The (multivariate) linear regression has the following form, where beta are the weighting parameters and epsilon is the

$$\hat{y}_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \epsilon_i \rightarrow \hat{\boldsymbol{y}}_i = \boldsymbol{x}_i^T \boldsymbol{\beta} + \boldsymbol{\epsilon}_i \rightarrow \hat{\boldsymbol{Y}} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

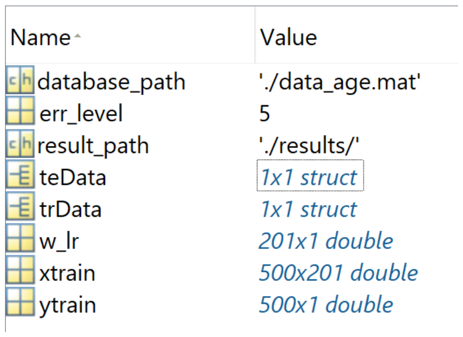
To optimize the weighting coefficients $\boldsymbol{\beta}$, least sum of squares error can be used between the ground truth and predicted labels. The following is the closed form formula for extracting the weighting coefficients using Sum of Least Squares Error. The aim is to find $\boldsymbol{\beta}$ that minimize: $SSE = (\boldsymbol{Y} - \hat{\boldsymbol{Y}})'(\boldsymbol{Y} - \hat{\boldsymbol{Y}})$

$$\boldsymbol{\beta} = (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{Y}$$

The MATLAB function *regress(…)*, calculates the weights of a multivariate regression model given feature matrix $\boldsymbol{X}$ and label vector pair $\boldsymbol{Y}$. The function can calculate confidence intervals for the weight coefficient estimates or estimate the weights for a given minimum confidence level using different parameter loadings. The function may also calculate different statistics, such as R^2, F-statistic and p-values for the weight estimates.

MATLAB documentation states that *regress(…)* is a simple implementation of multivariate *LinearModel* objects, which can produce preliminary results in a fast manner. As algorithm it uses Residual Intervals to estimate the weighting coefficients within a given confidence interval, which is 95% by default[2]. MATLAB documentation states that residuals for features have normal distributions with zero mean and with varying variances. The scale of the variances are normalized by "studentising" the residuals to a t-distribution with known degrees of freedom.

The code for this section can be found in the code listing; the following figure shows the output after running this code section.



Name	Value
database_path	'./data_age.mat'
err_level	5
result_path	'./results/'
teData	1x1 struct
trData	1x1 struct
w_lr	201x1 double
xtrain	500x201 double
ytrain	500x1 double

Figure 2. "w_lr" is the vector holding weighting coefficients of MLR model using Residual Intervals algorithm.

[2] More information can be found in the documentation: https://www.mathworks.com/help/stats/regress.html#:~:text=b%20%3D%20regress(%20y%20%2C%20X%20)%20returns%20a%20vector%20b,ones%20in%20the%20matrix%20X%20.

Question 3:

The test dataset can be read using the same way when reading training data. The MLR regression is defined by $\widehat{\boldsymbol{Y}} = \boldsymbol{X}\,\boldsymbol{\beta}.$ This can be realized easily by multiplying test features (of size 502 by 201) with the optimized weighting coefficients (of size 201 by 1) from Question 2. The output will a vector (of size 502 by 1) that has predictions for the individual feature samples. Code for this section can be found in the code listing. The predictions and ground truth labels for each test sample are given below:

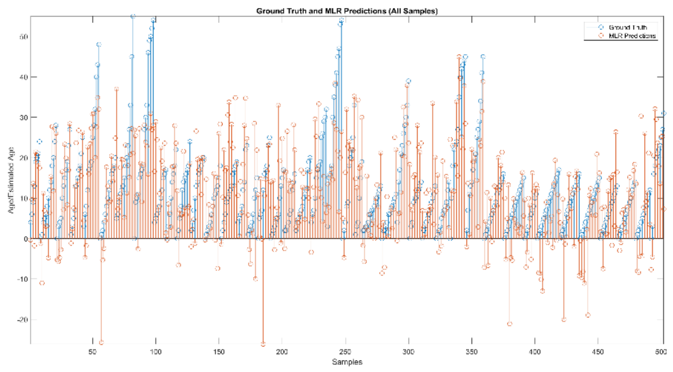


Figure 3. GT and MLR Predictions over all samples

Note that some of the values listed in the figure above is negative. These predictions indicate a negative age which is impossible due to the nature of the problem. This is normal as MLR fits a linear curve across all training samples; to minimize the residual loss. The model assumes linearity across the entire AAM domain and age range. If AAM representation of the sample corresponds to an age close to zero; the model may assume a negative age as it fits AAM feature vector linearly to the model. The following figure filters the negative age predictions and outputs the ground truths for these predictions. It is indeed the case, ground truth labels which are closer to zero are susceptible for such invalid predictions due to linear mapping between labels and AAM features.

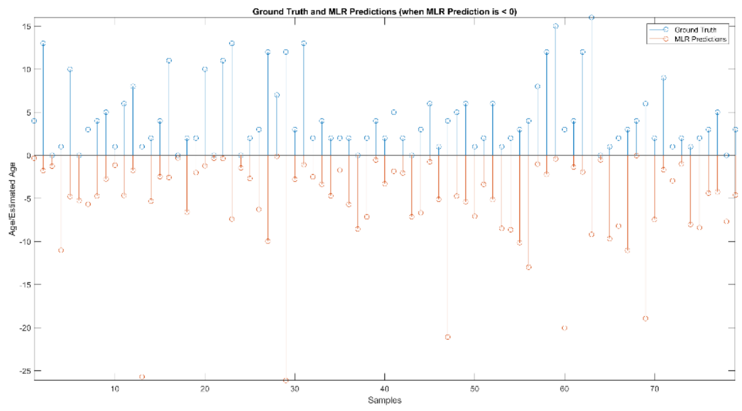


Figure 4. GT and MLR Predictions (where MLR prediction is less than 0)

Question 4:

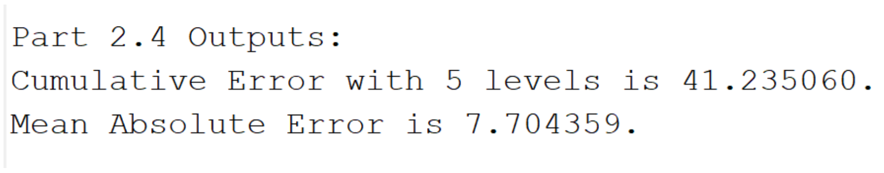
The Mean Absolute Error (MAE) has the following formula, where N is the total number of samples, y_k being the ground truth for kth sample and y_k being predicted value of kth sample:

$$MAE = \frac{1}{N}\sum_{k=1}^{N} |y_k - \widehat{y_k}|$$

Cumulative Score (CS) has the following formula, where N is the total number of samples and $\#_{e \le j}$ being the number of samples where the MAE of the estimated age with respect to the ground truth is less than j years:

$$CS(j) = \frac{1}{N}\#_{e \le j} \times 100$$

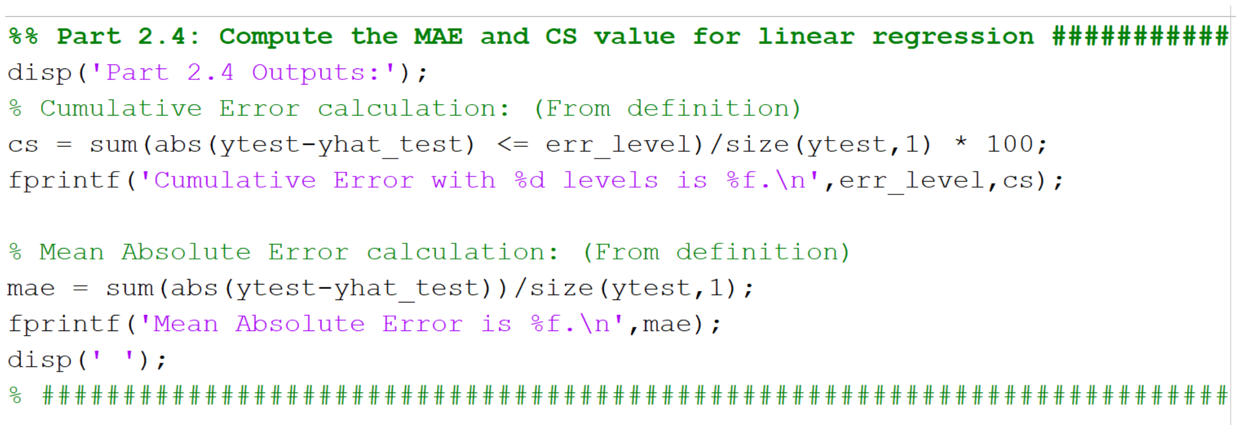
The code for this section can be found in the code listing. The following is the output required from this question.



```
Part 2.4 Outputs:
Cumulative Error with 5 levels is 41.235060.
Mean Absolute Error is 7.704359.
```

Figure 5. MAE and CS(5) metric outputs for MLR model.

For MLR model cumulative error with 5 levels is found to be 41.235060 and the mean absolute error is recorded as 7.704359. MAE indicate the average error in predictions of MLR model which is found to be 7.70 years. Cumulative Error with a level of 5 indicate that 41.2% of the predictions have an error less than 5 years.



```
%% Part 2.4: Compute the MAE and CS value for linear regression ###########
disp('Part 2.4 Outputs:');
% Cumulative Error calculation: (From definition)
cs = sum(abs(ytest-yhat_test) <= err_level)/size(ytest,1) * 100;
fprintf('Cumulative Error with %d levels is %f.\n',err_level,cs);

% Mean Absolute Error calculation: (From definition)
mae = sum(abs(ytest-yhat_test))/size(ytest,1);
fprintf('Mean Absolute Error is %f.\n',mae);
disp(' ');
% ##########################################################################
```

Figure 6. Code Listing for Question 4

Question 5:

This question loops the calculation of CS multiple times to generate the plot for CS against the cumulative error level. The code for this section can be found in the code listing, at the end of this report.

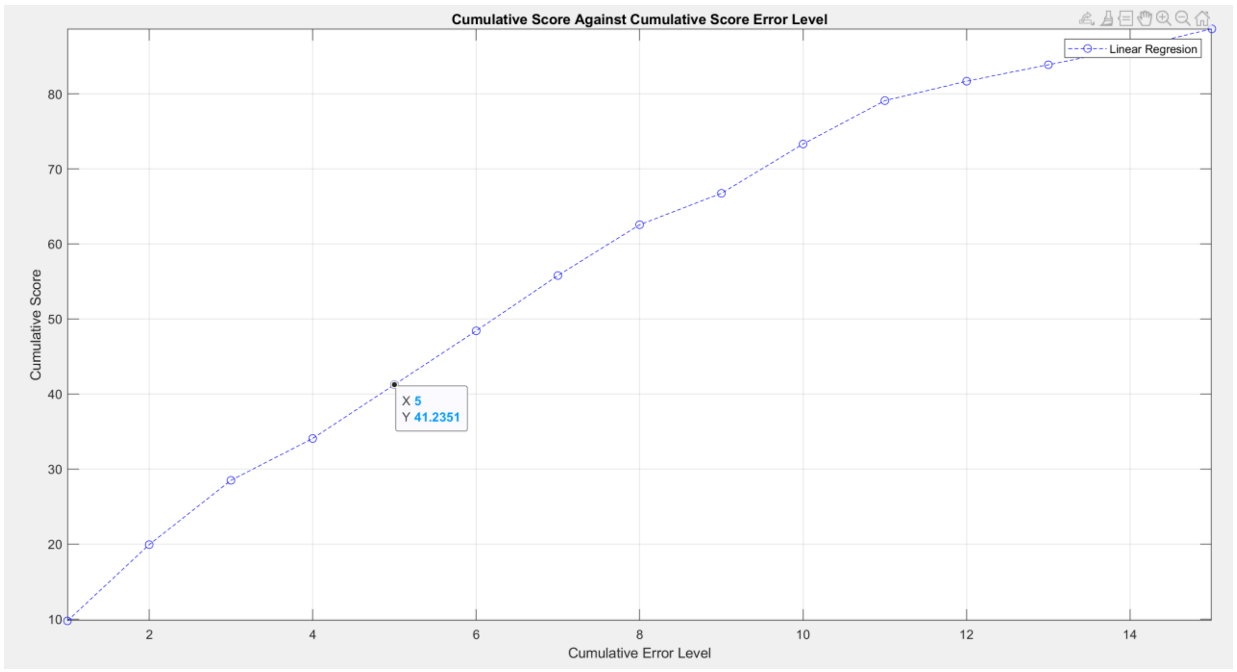
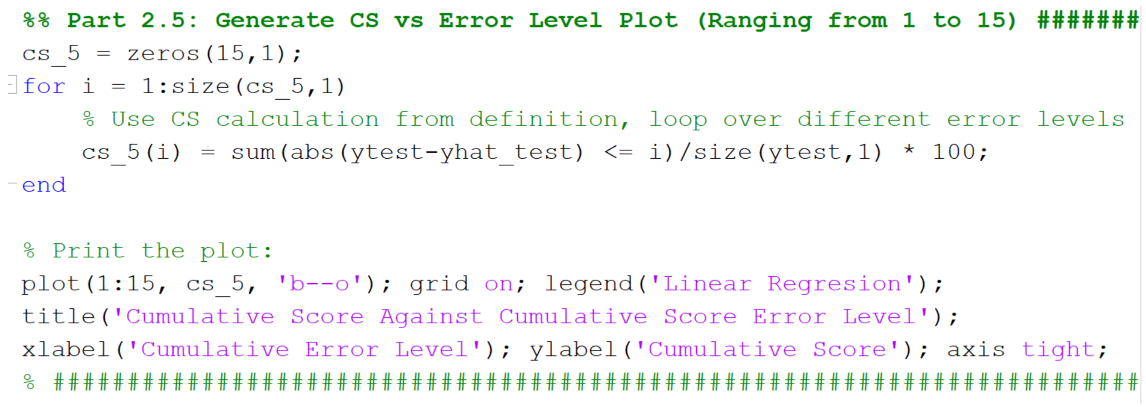


Figure 7. Cumulative Score Against Cumulative Score Error Level for MLR Model

It is seen that the output for CS(5) holds true from Question 4. As levels increase, the cumulative score approaches to 1, as the margin of error acceptance increases, increasing $\#_{e \leq j}$. At an error level of 69 the metric will reach to 1; as all samples are counted by $\#_{e \leq j}$, as error level is equivalent to the maximum age value that the samples can get. For example, ground truth being 69 and prediction being 0 would be counted towards $\#_{e \leq j}$ if error level is set to 69.



```
%% Part 2.5: Generate CS vs Error Level Plot (Ranging from 1 to 15) #######
cs_5 = zeros(15,1);
for i = 1:size(cs_5,1)
    % Use CS calculation from definition, loop over different error levels
    cs_5(i) = sum(abs(ytest-yhat_test) <= i)/size(ytest,1) * 100;
end

% Print the plot:
plot(1:15, cs_5, 'b--o'); grid on; legend('Linear Regresion');
title('Cumulative Score Against Cumulative Score Error Level');
xlabel('Cumulative Error Level'); ylabel('Cumulative Score'); axis tight;
% ########################################################################
```

Figure 8. Code Listing for Question 5

Question 6:

This question will individually implement Partial Least Squares regression model and Regression Tree models using MATLAB's built-in functions and will output MAE and CS with a level of 5.

Partial Least Squares (PLS) Regression method, that is like MLR and PCA[3]. Instead of using feature parameters in MLR, it deconstructs the original feature and label space to loading matrices[4]. The formulation is listed below:

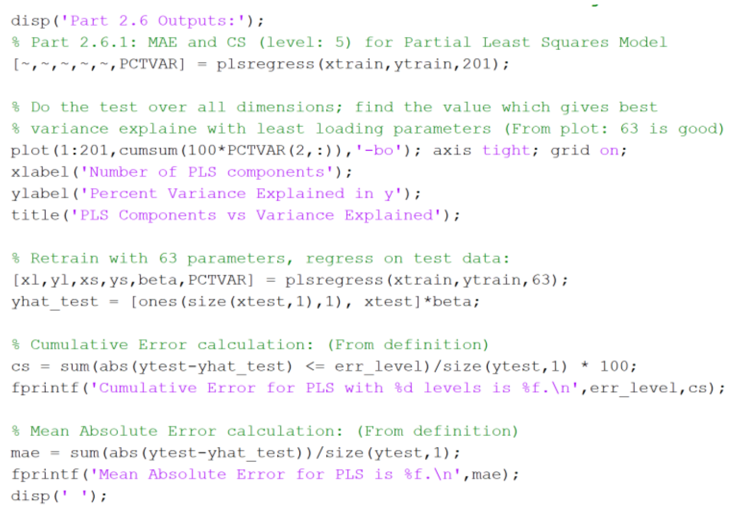
$$X = TP^T + E$$

$$Y = UQ^T + F$$

Here, X (N by 201) and Y (N by 1) are AAM features and age labels respectively. T and U are the projection matrices; E and F are error matrices that are i.i.d. random normal variables. P and Q are the loading matrices, that can be used with projection matrices to reconstruct the original feature and label space. The objective of PLS is to decompose X and Y jointly to maximize the covariance between projection matrices T and U. In this process PLS components selected to be uncorrelated.

MATLAB has a built-in function called as *plsregress(...)* which can construct PLS models. The documentation states that this method uses *SIMPLS* algorithm which first standardizes the AAM features and age labels, then uses SVD to decompose the feature and label spaces[5].

The following code segment was used to construct a PLS model:



```
disp('Part 2.6 Outputs:');
% Part 2.6.1: MAE and CS (level: 5) for Partial Least Squares Model
[~,~,~,~,~,PCTVAR] = plsregress(xtrain,ytrain,201);

% Do the test over all dimensions; find the value which gives best
% variance explaine with least loading parameters (From plot: 63 is good)
plot(1:201,cumsum(100*PCTVAR(2,:)),'-bo'); axis tight; grid on;
xlabel('Number of PLS components');
ylabel('Percent Variance Explained in y');
title('PLS Components vs Variance Explained');

% Retrain with 63 parameters, regress on test data:
[xl,yl,xs,ys,beta,PCTVAR] = plsregress(xtrain,ytrain,63);
yhat_test = [ones(size(xtest,1),1), xtest]*beta;

% Cumulative Error calculation: (From definition)
cs = sum(abs(ytest-yhat_test) <= err_level)/size(ytest,1) * 100;
fprintf('Cumulative Error for PLS with %d levels is %f.\n',err_level,cs);

% Mean Absolute Error calculation: (From definition)
mae = sum(abs(ytest-yhat_test))/size(ytest,1);
fprintf('Mean Absolute Error for PLS is %f.\n',mae);
disp(' ');
```

Figure 9. Code Listing for Question 6, Part 1.

First *plsregress(...)* function was used with 201 components (maximum amount given original dimensionality of data) to evaluate how much variance is explained by each component. Note that the bias parameter is also added by using *ones* method in the code listing above.

[3] https://www.mathworks.com/help/stats/partial-least-squares.html
[4] https://en.wikipedia.org/wiki/Partial_least_squares_regression
[5] See footnote 3.

The following figure shows the plot for variance explained by the components:

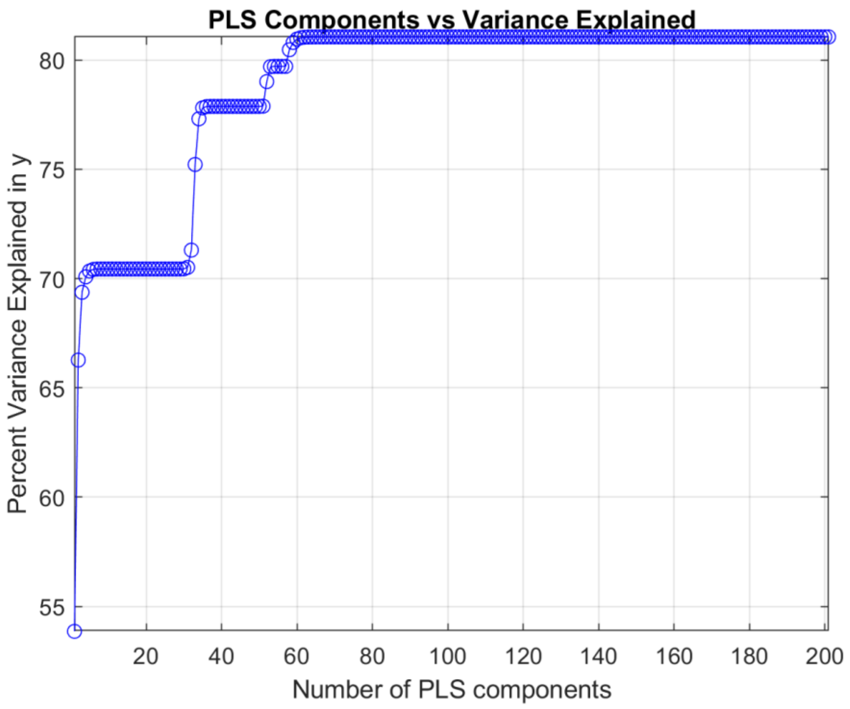


Figure 10. PLS Components vs Variance Explained in Y.

From the figure, it is seen that 63 components were sufficient to explain over 80% of the variance in the age predictions. Unlike PCA components, PLS components do not have to be orthonormal, the only restriction for PLS components is the components being uncorrelated with respect to each other. Therefore, unlike PCA, all PLS components do not explain 100 % of the variance in Y.

Using the number of parameters 10, 38 and 63, the best results were achieved with 10 PLS parameters. This could be because that lower number of PLS parameters explain the signal more and the noise less; therefore, giving better results. For 63 parameters, MAE and CS(5) was 7.70 and 40.03% respectively.

After finding the smallest number of components that explain the largest variance, the model is trained once again and the predictions are extracted by using weighting coefficients of PLS model, namely *beta*. The following MAE and CS(5) values were extracted. 53% of the samples have less than MAE (absolute age difference) of 5 years, whereas the average error is found to be 6.07 years.

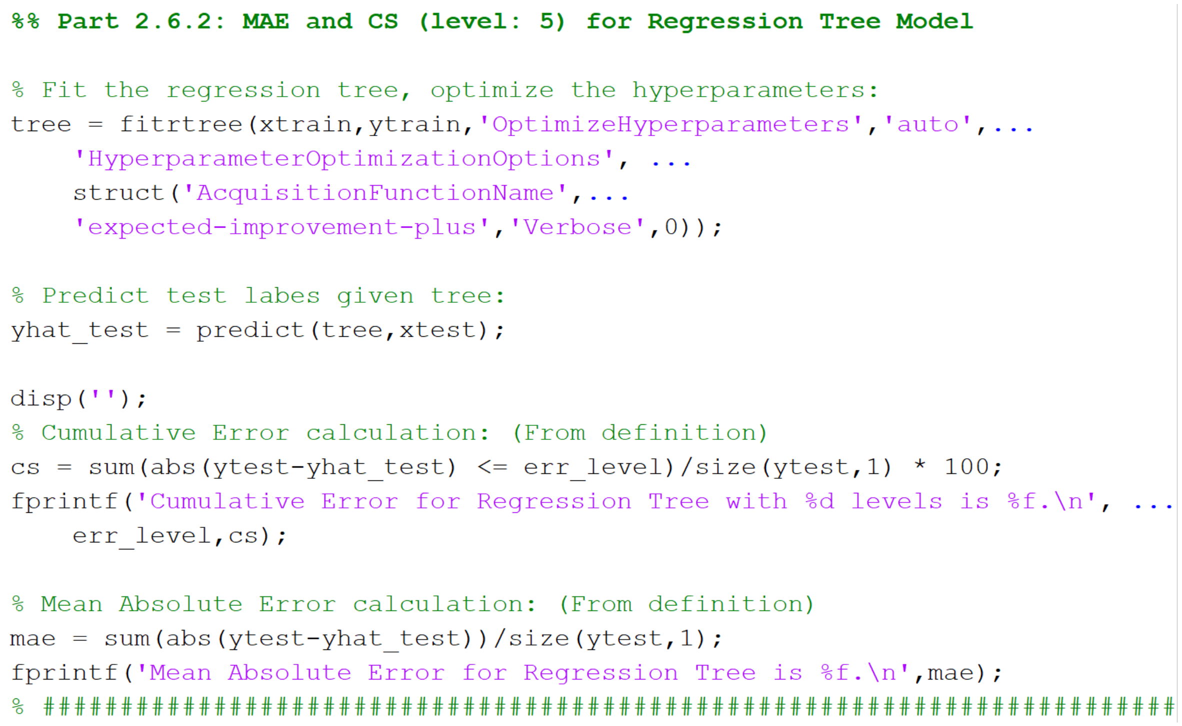
```
Part 2.6 Outputs:
Cumulative Error for PLS with 5 levels is 52.589641.
Mean Absolute Error for PLS is 6.070294.
```

Figure 11. MAE and CS(5) outputs for PLS Model.

Regression trees are used when the predictions are continuous variables. Many metrics such as Gini impurity or information gain can be used to split the dataset into regression classes[6]. MATLAB has built in function called *fitrtree(...)* to construct and optimize regression trees automatically using training data and training labels (AAM features and corresponding age labels).

Node splitting is done using standard CART algorithm. MSE error between ground truth and predictions are calculated. The splits are made to masximize the reduction in MSE error. More details can be found in MATLAB documentations[7].

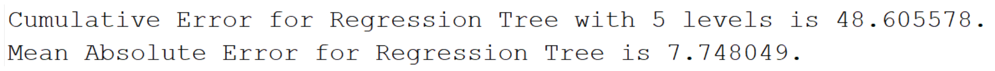
The code listing for the regression tree implementation is given below:



```
%% Part 2.6.2: MAE and CS (level: 5) for Regression Tree Model

% Fit the regression tree, optimize the hyperparameters:
tree = fitrtree(xtrain,ytrain,'OptimizeHyperparameters','auto',...
    'HyperparameterOptimizationOptions', ...
    struct('AcquisitionFunctionName',...
    'expected-improvement-plus','Verbose',0));

% Predict test labes given tree:
yhat_test = predict(tree,xtest);

disp('');
% Cumulative Error calculation: (From definition)
cs = sum(abs(ytest-yhat_test) <= err_level)/size(ytest,1) * 100;
fprintf('Cumulative Error for Regression Tree with %d levels is %f.\n', ...
    err_level,cs);

% Mean Absolute Error calculation: (From definition)
mae = sum(abs(ytest-yhat_test))/size(ytest,1);
fprintf('Mean Absolute Error for Regression Tree is %f.\n',mae);
% ########################################################################
```

Figure 12. Code Listing for Question 6, Part 2.

The given listing directly uses training and testing data to construct a regression tree. By default Bayesian Optimization is used within CART algorithm which is implemented within *fitrtree(...)*. The optimization is done automatically by feeding *OptimizeHyperparameters* setting. *Verbose* is set to 0 to omit intermediate steps. The following are the MAE and CS(5) results obtained from regression tree.



```
Cumulative Error for Regression Tree with 5 levels is 48.605578.
Mean Absolute Error for Regression Tree is 7.748049.
```

Figure 13. MAE and CS(5) outputs for Regression Tree Model.

[6] For more information see: https://en.wikipedia.org/wiki/Decision_tree_learning#Further_reading
[7] For more information see: https://www.mathworks.com/help/stats/fitrtree.html#bunrdhe-1

The following figures are produced by the optimization process, listed below for completeness.

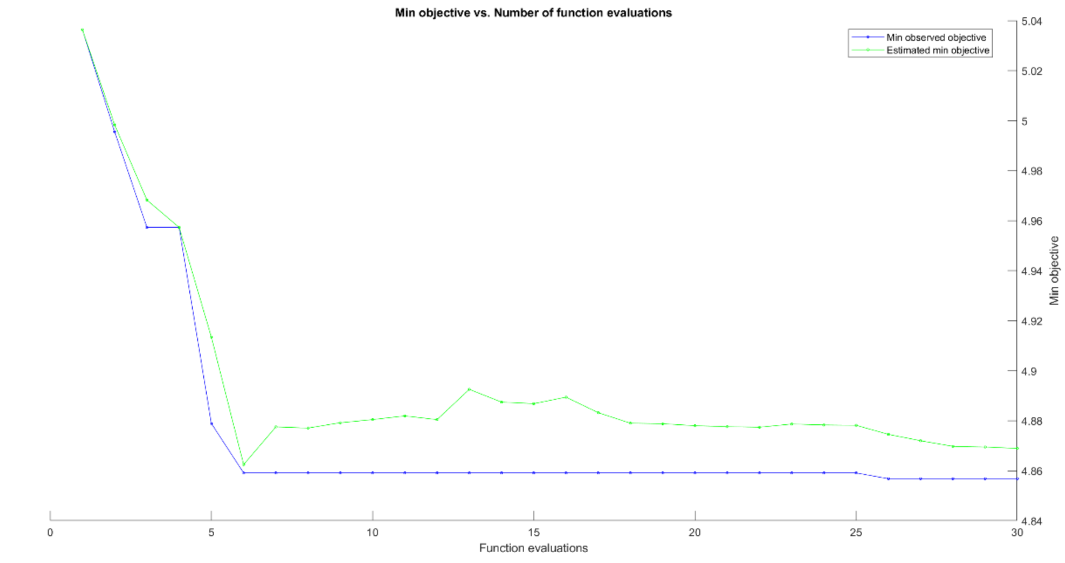


Figure 14. Regression Tree Objective Minimization Process.

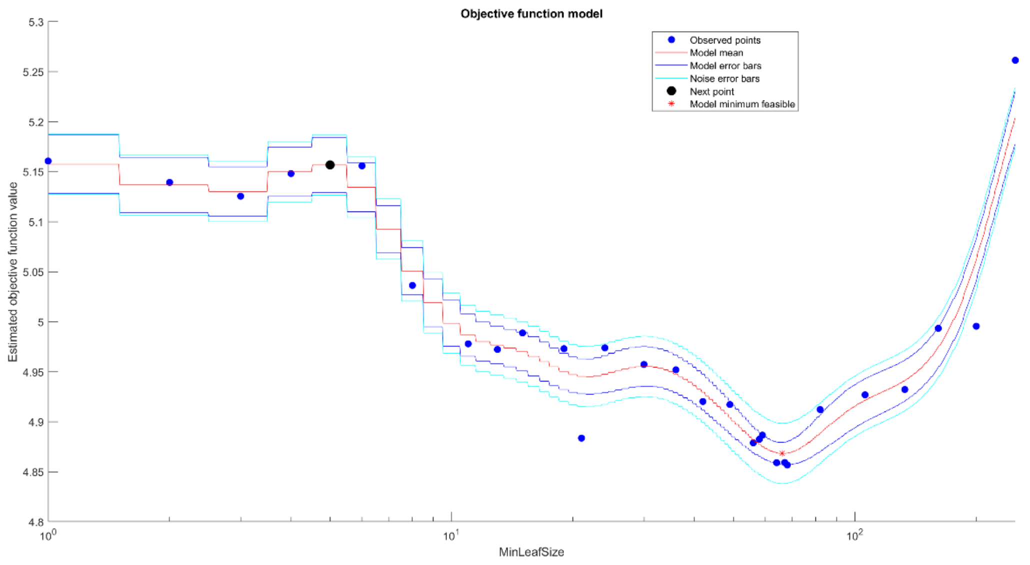


Figure 15. Objective Function Model with respect to Minimum Leaf Size

Note that Regression tree have 0.05 more MAE score than MLR and PLS methods. Although the average error in predictions over all samples are larger, the cumulative score with a level of 5 is 7% and 8% better compared to MLR and PLS, respectively. This means that the more samples have a MAE lower than 5, while the outlier errors are much larger than MLR and PLS, moving the average error higher.

Question 7:

Support Vector Regression (SVR) is very similar to Support Vector Machines (SVM). In SVMs the aim is to find the best hyperplane that separates all training samples with the highest separation between class boundaries. This is achieved by minimizing the weighting coefficients of the regression model, which maximizes the the margin between the ground truth and mode predictions within an error, defined by epsilon. The minimization and the constraint are as follows:

$$\min \frac{1}{2}||w||^2$$

$$|\boldsymbol{Y} - \boldsymbol{X}\,\boldsymbol{\beta}| \leq \varepsilon$$

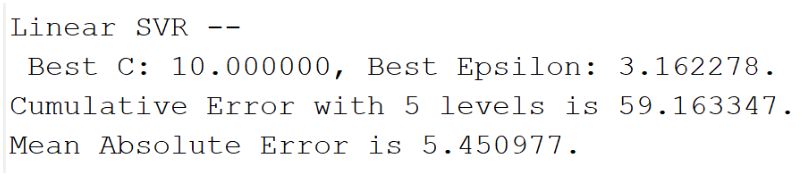
The epsilon formulation creates a threshold for the predictions, such that all predictions must be within epsilon range to the ground truth. In the training dataset, data that are epsilon away from the regression model will not be utilized in the training process. Additionally, a slack variable could be added to the minimization cost, which takes into account the data which are xi away from the support vectors. The updated minimization and constraints are as follows:

$$\min \frac{1}{2}||w||^2 + C\sum_{i=1}^{n} |\xi|$$

$$|\boldsymbol{Y} - \boldsymbol{X}\,\boldsymbol{\beta}| \leq \varepsilon + |\xi|$$

Finally, SVR can use many different kernels to regress given dataset, some examples being, linear, polynomial, and radial basis function kernels. For this assignment LibSVM 3.14 was used to build SVR models[8]. It is downloaded from the footnote given below and compiled for MATLAB. As for the options, ***Epsilon-SVR (-s 3)*** was selected. More information to Epsilon-SVR's implementation can be found in this link[9]. Epsilon-SVR provides more fine-tuning capability over nu-SVR.

A grid search over different kernel types can be used. For linear kernels grid search could be done with respect to epsilon and C values. For polynomial, radial basis and sigmoid kernels, an additional search on gamma can be done. The hyperparameters selected for the search can be found in the code listing. The search space is coarsely sampled, ranging between plausible values for epsilon, C, and gamma.

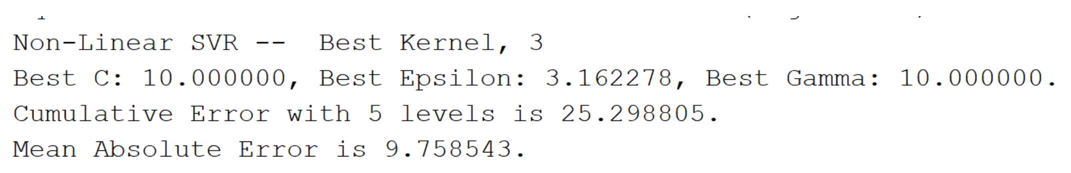


```
Linear SVR --
 Best C: 10.000000, Best Epsilon: 3.162278.
Cumulative Error with 5 levels is 59.163347.
Mean Absolute Error is 5.450977.
```

Figure 16. Best Results for E-SVR with Linear Kernel.

[8] For more information see: https://www.csie.ntu.edu.tw/~cjlin/libsvm/
[9] E-SVR Implementation Details: https://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf

The following results do grid search over gamma, epsilon, and C values; as polynomial, radial basis and sigmoid kernels also have gamma to optimize over. It returns the best predicted score.



```
Non-Linear SVR --  Best Kernel, 3
Best C: 10.000000, Best Epsilon: 3.162278, Best Gamma: 10.000000.
Cumulative Error with 5 levels is 25.298805.
Mean Absolute Error is 9.758543.
```

Figure 17. 3D grid search on Non-linear kernels. Best result with sigmoid kernels.

E-SVR with non-linear kernels gave significantly lower results than E-SVR with linear kernels. In this search the best kernel is found to be sigmoid kernels with the hyperparameters tested above. It is seen that the best values for C and gamma is found at the edge scenarios of the grid search, implying widening the search space may improve the results.

SVR models are highly sensitive to the hyperparameters, therefore using more refined search may improve the overall scores and search process. However, working through such search space is computationally expensive, therefore doing larger, more refined experiments are not plausible. Linear kernels have lesser parameters to optimize, hence the optimization is easier to handle. The performance of E-SVR with linear kernels is comparable to other methods if not better.

Appendix 1 - Comparison of MAE and Cumulative Scores of MLR, PLS-R, Regression Tree and SVR Models:

Table 1. Comparison between scores for different Regression Models

Model \ Score Type	**Mean Abs. Error (MAE)**	**Cumulative Score (5)**
Multivariate Linear Regr.	7.704359	41.235060
Partial LS Regr. (10 comp)	**6.070294**	**52.589641**
Regression Tree	7.748049	48.605578
E-SVR. (Linear Kernel)	**5.450977**	**59.163347**
E-SVR (Sigmoid Kernel)	9.758543	25.298805

From the table above the best Model is found to be optimized E-SVR with a linear kernel, followed by PLS Regression. Both models have more than 50% of the predictions having less than 5 years error, compared to their ground truth.

E-SVR with non-linear kernels provided the worst performance, however it was observed that the "optimal parameters" of the grid search was found in the corner cases; implying a wider search may benefit the scores more. SVR is highly sensitive to parameter changes; therefore, a more refined search may have improved the accuracy. However, the linear kernel E-SVR is simpler and provides satisfactory performance compared to non-linear methods; being the better alternative to non-linear models.

PLS Regression uses a small subset of projected loading vectors to summarize information in a smaller domain. PLS Regression have similarities to MLR and PCA in the sense of how it is trained and how it reduces the number of dimensions. A lower subset of PLS components proved to be better at ignoring noise in AAM features; improving both MAE and CS(5) scores significantly.

Regression tree also improves over MLR model, however certain AAM features have significantly big errors in between predicted ages and ground truth ages. While approximately 50% of the samples had a prediction error less than 5 years; mean error is larger than MLR.

Age estimation is a regression problem. Given AAM features of human faces, these models estimate an age for an unknown individual. It is very hard to pinpoint an exact age to an individual. MAE can be used to calculate the error between individual's age and the regressed age. However, MAE is not sufficient by itself as it only states the mean prediction error across all samples. Most of the faces have different facial features, poses, expressions or lighting conditions that might affect the age estimation. Cumulative score becomes a complimentary metric to MAE (internally using MAE within itself), by stating given a threshold on the estimation error that an algorithm can make, what is percentage of samples that are within the error threshold that is deemed acceptable. Higher the cumulative score at lower error levels, less prediction error is tolerated, and the percentage of the samples that are within accepted limits of error are measured. Along with MAE, cumulative score can be used as a metric to assess model performance in estimating age of an individual.

Appendix 2 - Full Code Listing:

Full code listing can be found (in text format) below. Note that individual questions and parts are divided by comments. Comments on the code follow from the discussions in the report.

```
% *************************************************************************
% ECS797P - Machine Learning for Visual Data Analytics
% Assignment 3 - Facial Age Estimation by (Linear) Regression
% Hasan Emre Erdemoglu - 200377106, Due: April 1st 2021
% Acknowledgements: *******************************************************
% libsvm library is used in this assignment.
% LibSVM library:      http://www.csie.ntu.edu.tw/~cjlin/libsvm/
addpath(genpath('libsvm')); % add the path of code to your workspace
% libsvm has to be compiled via this comment when necessary:
% run('/libsvm/matlab/make')
% *************************************************************************
%% Part 1: Getting Started -- Settings:
clear; clc; % For debugging purposes
database_path = './data_age.mat';

% Cumulative error level setting: (Global)
err_level = 5;

%% Part 2.1 - 2.2: Read train/test data and regress() ####################
load(database_path); % Part 2.1, loads the dataset.

xtrain = trData.feat; % Features pulled from training data struct
ytrain = trData.label; % Labels pulled from training data struct
w_lr = regress(ytrain,xtrain); % Use Linear Regr. from MATLAB fcns.
% #########################################################################

%% Part 2.3: Read testing data, apply learned linear regression ###########
xtest = teData.feat; % Features pulled from testing data struct
ytest = teData.label; % Labels pulled from testing data struct
yhat_test = xtest * w_lr; % Use learned weights from training to predict

% Plot graph to show the negative age predictions, and when they happen:
neg_preds_idx = yhat_test < 0; % Filter negative predictions
figure; stem(ytest(neg_preds_idx == 1)); hold on;
stem(yhat_test(neg_preds_idx == 1));
title('Ground Truth and MLR Predictions (when MLR Prediction is < 0)');
xlabel('Samples'); ylabel('Age/Estimated Age'); axis tight;
legend('Ground Truth', 'MLR Predictions');

figure; stem(ytest); hold on; stem(yhat_test);
title('Ground Truth and MLR Predictions (All Samples)');
xlabel('Samples'); ylabel('Age/Estimated Age'); axis tight;
legend('Ground Truth', 'MLR Predictions');
% #########################################################################

%% Part 2.4: Compute the MAE and CS value for linear regression ###########
disp('Part 2.4 Outputs:');
% Cumulative Error calculation: (From definition)
cs = sum(abs(ytest-yhat_test) <= err_level)/size(ytest,1) * 100;
fprintf('Cumulative Error with %d levels is %f.\n',err_level,cs);

% Mean Absolute Error calculation: (From definition)
mae = sum(abs(ytest-yhat_test))/size(ytest,1);
fprintf('Mean Absolute Error is %f.\n',mae);
disp(' ');
% #########################################################################

%% Part 2.5: Generate CS vs Error Level Plot (Ranging from 1 to 15) #######
cs_5 = zeros(15,1);
for i = 1:size(cs_5,1)
    % Use CS calculation from definition, loop over different error levels
    cs_5(i) = sum(abs(ytest-yhat_test) <= i)/size(ytest,1) * 100;
end
```

```
% Print the plot:
figure; plot(1:15, cs_5, 'b--o'); grid on; legend('Linear Regresion');
title('Cumulative Score Against Cumulative Score Error Level');
xlabel('Cumulative Error Level'); ylabel('Cumulative Score'); axis tight;
% ##########################################################################

%% Part 2.6.1: Use MATLAB built-in functions to do following: #############
disp('Part 2.6 Outputs:');
% Part 2.6.1: MAE and CS (level: 5) for Partial Least Squares Model
[~,~,~,~,~,PCTVAR] = plsregress(xtrain,ytrain,201);

% Do the test over all dimensions; find the value which gives best
% variance explaine with least loading parameters (From plot: 10 is good)
figure; plot(1:201,cumsum(100*PCTVAR(2,:)),'-bo'); axis tight; grid on;
xlabel('Number of PLS components');
ylabel('Percent Variance Explained in y');
title('PLS Components vs Variance Explained');

% Retrain with 63 parameters, regress on test data:
[xl,yl,xs,ys,beta,PCTVAR] = plsregress(xtrain,ytrain,10);
yhat_test = [ones(size(xtest,1),1), xtest]*beta;

% Cumulative Error calculation: (From definition)
cs = sum(abs(ytest-yhat_test) <= err_level)/size(ytest,1) * 100;
fprintf('Cumulative Error for PLS with %d levels is %f.\n',err_level,cs);

% Mean Absolute Error calculation: (From definition)
mae = sum(abs(ytest-yhat_test))/size(ytest,1);
fprintf('Mean Absolute Error for PLS is %f.\n',mae);
disp(' ');

%% Part 2.6.2: MAE and CS (level: 5) for Regression Tree Model
% Fit the regression tree, optimize the hyperparameters:
tree = fitrtree(xtrain,ytrain,'OptimizeHyperparameters','auto',...
    'HyperparameterOptimizationOptions', ...
    struct('AcquisitionFunctionName',...
    'expected-improvement-plus','Verbose',0));

% Predict test labes given tree:
yhat_test = predict(tree,xtest);

disp('');
% Cumulative Error calculation: (From definition)
cs = sum(abs(ytest-yhat_test) <= err_level)/size(ytest,1) * 100;
fprintf('Cumulative Error for Regression Tree with %d levels is %f.\n', ...
    err_level,cs);

% Mean Absolute Error calculation: (From definition)
mae = sum(abs(ytest-yhat_test))/size(ytest,1);
fprintf('Mean Absolute Error for Regression Tree is %f.\n',mae);
% ##########################################################################

%% Part 2.7: MAE & CS value (level: 5) for SV Regression by using LIBSVM ##
% Selecting epsilon-SVR(s=3), kernel type, cost and gamma of kernel fcn is
% selected by 5-fold cross-validation.
bestcv=inf; bestc = 0; bestep = 0;
disp('Part 2.7 Outputs:');
% Using Linear Kernel (No search in Gamma values):
i = 0;
for log10c = -1:0.5:2 % Sensible range of C values (7 tests)
    for log10ep = -2:0.5:1 % Sensible range of epsilon values (7 tests)
        cmd = ['-s 3 -v 5 -t 0 -c ', num2str(10^log10c),' -p ', ...
            num2str(10^log10ep), ' -b 1 -q'];
        cv = svmtrain(ytrain, xtrain, cmd);
        if (cv <= bestcv)
            bestcv = cv; bestc = 10^log10c; bestep = 10^log10ep;
        end
        i=i+1;
        fprintf('Iteration %d. ', i);
        fprintf('Epsilon: %d, C: %d .\n', 10^log10ep, 10^log10c);
        fprintf( 'Best C: %f, Best Epsilon: %f. \n\n', bestc, bestep);
        disp('');
    end
end

% Set the options and retrain best found model:
options = sprintf('-s 3 -t 0 -c %f -p %f -q', bestc, bestep);
model=svmtrain(ytrain, xtrain, options);
[ytest_hat, ~ , ~] = svmpredict(ytest, xtest, model);

% Set the options and retrain best found model:
% options = sprintf('-s 3 -t %d -c %f -g %f -b 1 -q', 1, 1/201, 1);

% Compute MAE and CS (with 5 levels):
disp('Part 2.7 Outputs:');
% Cumulative Error calculation:
fprintf( 'Linear SVR --\nBest C: %f, Best Epsilon: %f. \n', bestc, bestep);
cs = sum(abs(ytest-ytest_hat) <= err_level)/size(ytest,1) * 100;
fprintf('Cumulative Error with %d levels is %f.\n',err_level,cs);

% Mean Absolute Error calculation:
mae = sum(abs(ytest-ytest_hat))/size(ytest,1);
fprintf('Mean Absolute Error is %f.\n',mae);
disp(' ');

%% Non-linear kernels:
% All the tests use 5-fold CV (-v 5), using silent mode (-q) and
% probability estimates (-b 1)
% Using non-linear kernels (Search space in kernel, gamma, c, epsilon):
i = 0; bestcv=inf; bestc = 0; bestg = 0; bestep = 0; bestk = 0;
for kernel = 1:3 % Remaining kernel types (3 tests)
    for log10c = -1:1:2 % Sensible range of C values (4 tests)
        for log10ep = -2:0.5:1 % Sensible range of epsilon values (7 tests)
            for log10g = -3:1:1 % Sensbl. range of gamma values (5 tests)
                cmd = ['-s 3 -v 5 -t ',kernel, ' -c ', num2str(10^log10c), ...
                    ' -p ',num2str(10^log10ep), ' -g ', log10g ' -b 1 -q'];
                cv = svmtrain(ytrain, xtrain, cmd);
                if (cv <= bestcv)
                    bestcv = cv; bestc = 10^log10c; bestk = kernel;
                    bestep = 10^log10ep; bestg = 10^log10g;
                end
                i=i+1;
                fprintf('Iteration %d. ', i);
                fprintf('Gamma: %d, kernel: %d. \n', 10^log10g, kernel);
                fprintf('Epsilon: %d, C: %d .\n', 10^log10ep, 10^log10c);
                fprintf( 'Best C: %f, Best Epsilon: %f. \n', bestc, bestep);
                fprintf('Best Gamma: %d, best kernel: %d. \nn', 10^log10g, bestk);
                disp('');
            end
        end
    end
end

options = sprintf('-s 3 -t %f -c %f -p %f -q -g %f', ...
    bestk, bestc, bestep, bestg);
model=svmtrain(ytrain, xtrain, options);
[ytest_hat, ~ , ~] = svmpredict(ytest, xtest, model);

% Compute MAE and CS (with 5 levels):
% Cumulative Error calculation:
fprintf( 'Non-Linear SVR --  Best Kernel, %d \n', bestk);
fprintf('Best C: %f, Best Epsilon: %f, Best Gamma: %f. \n', ...
    bestc, bestep, bestg);
cs = sum(abs(ytest-ytest_hat) <= err_level)/size(ytest,1) * 100;
fprintf('Cumulative Error with %d levels is %f.\n',err_level,cs);

% Mean Absolute Error calculation:
mae = sum(abs(ytest-ytest_hat))/size(ytest,1);
fprintf('Mean Absolute Error is %f.\n',mae);
disp(' ');
% #########################################################################
```