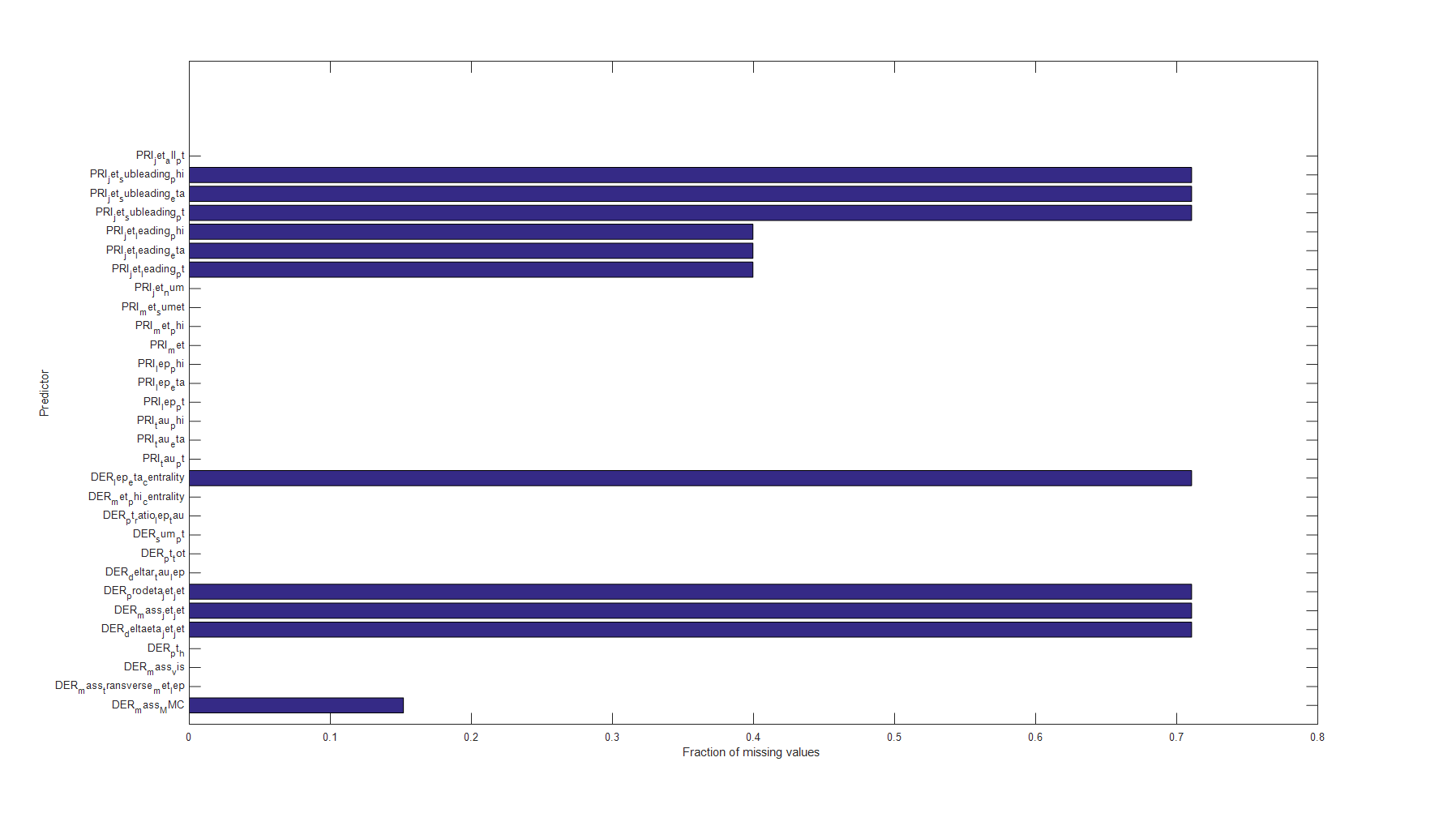
# **Missing values**



% The diagram shows the fraction of missing values in each feature (predictor)

% column\_names(1) = [];

% column\_names(end) = [];

% figure;

% barh(sum(isnan(raw\_data), 1) / size(raw\_data, 1));

% h = gca;

% h.YTick = 1:numel(column\_names);

% h.YTickLabel = column\_names;

% ylabel 'Predictor';

% xlabel 'Fraction of missing values';

1. The missing values are imputed by using ‘knnimpute’ function with default parameters. First, the data set is sorted by ‘eventID’ in ascending order, in order to get better result in imputation step. -999 is replaced with NaN and it is executed ‘knnimpute’. The result is scaled in order to have zero mean and unitary standart deviation.
2. Also, it is used PCA using the alternating least squares (ALS) algorithm. The ALS algorithm estimates the missing values in the data. The results are below:

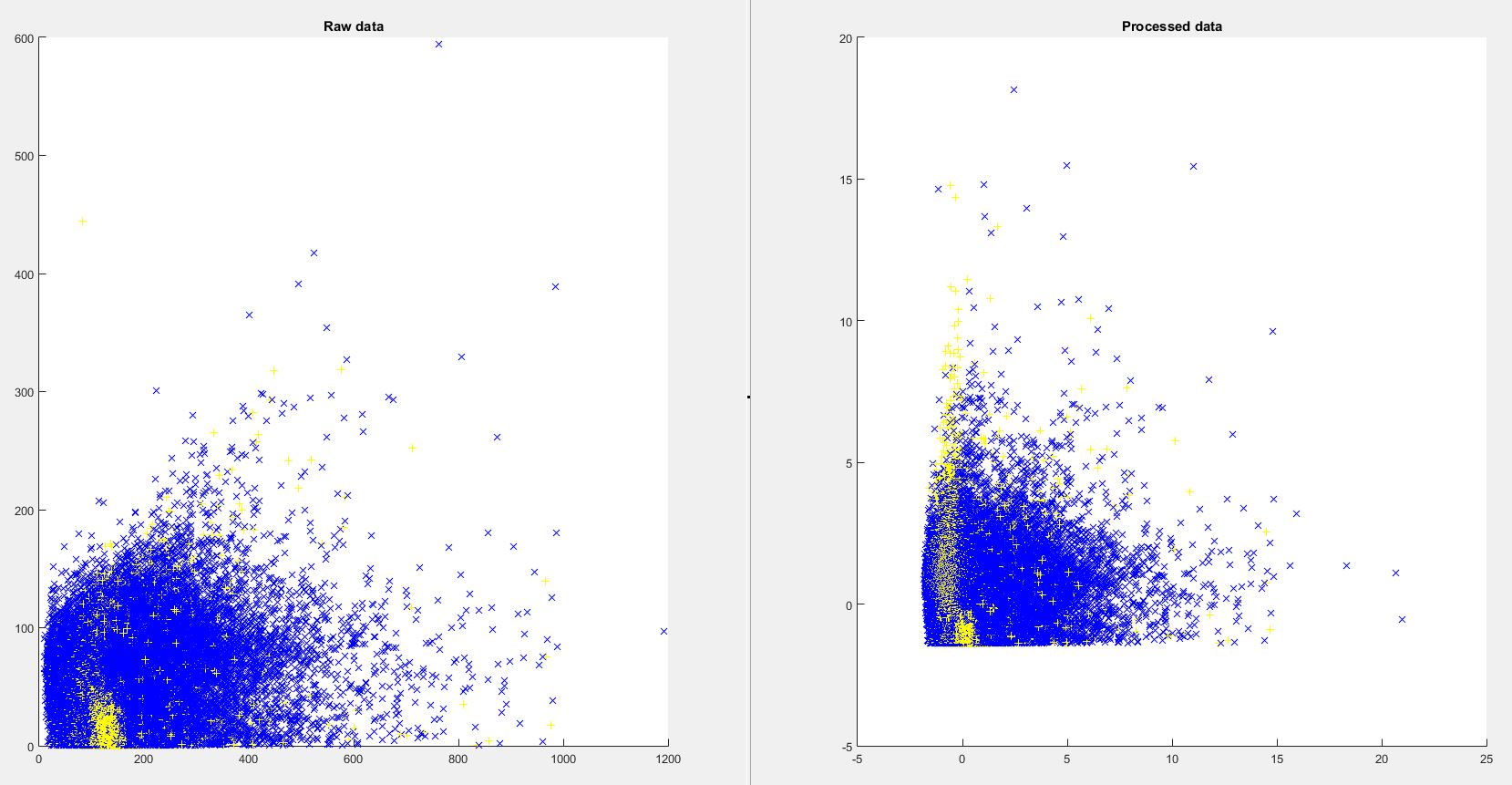


Figure 1 Plotting the data before and after preprocessing (with knnimpute)

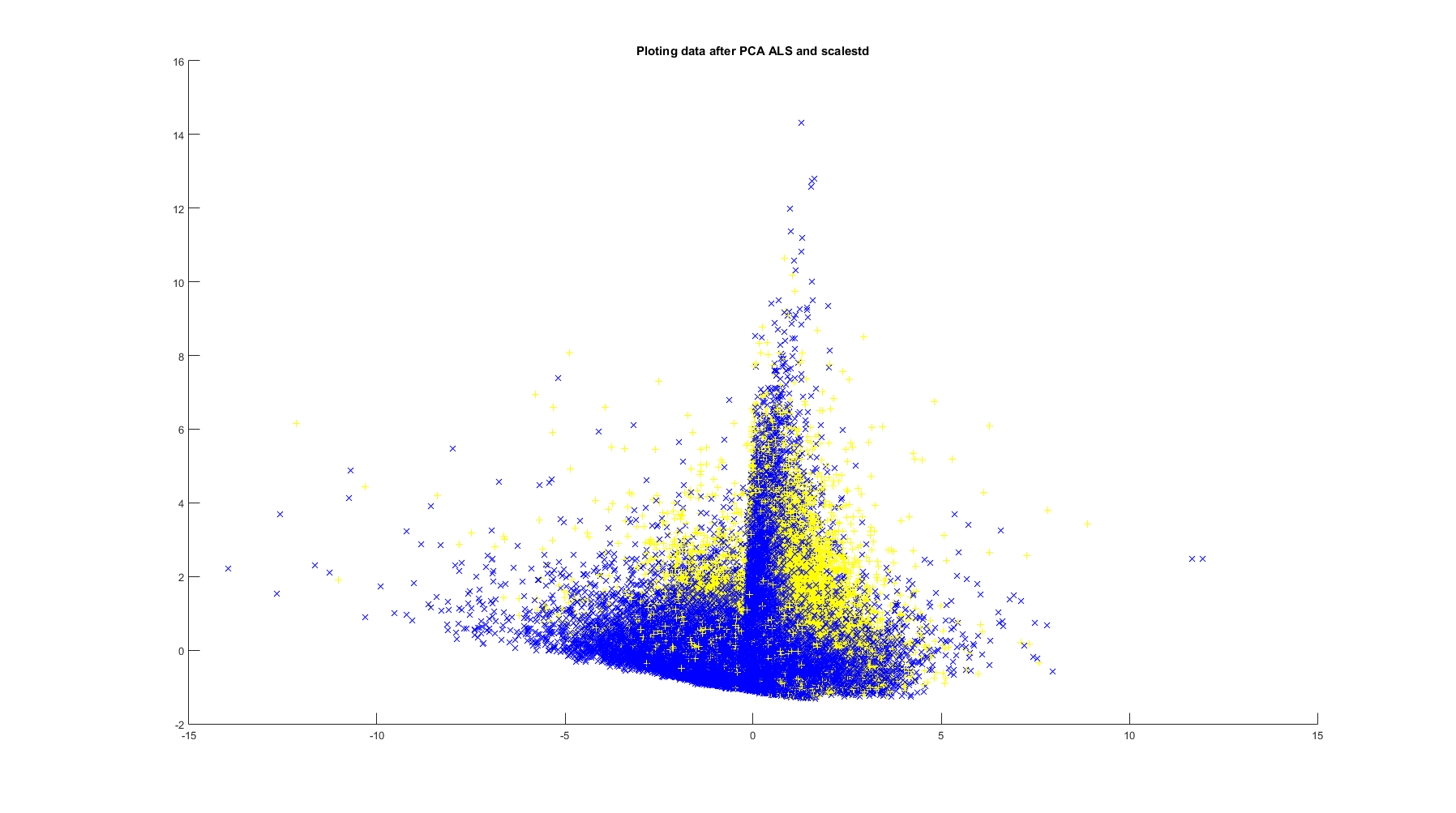


Figure 2Plotting of the data after PCA ALS

1. The missing values are imputed with ‘inpaint\_nans’, as well. It uses a spring metaphor. Assumes springs (with a nominal length of zero) connect each node with every neighbor (horizontally, vertically and diagonally). Since each node tries to be like its neighbors, extrapolation is as a constant function where this is consistent with the neighboring nodes.

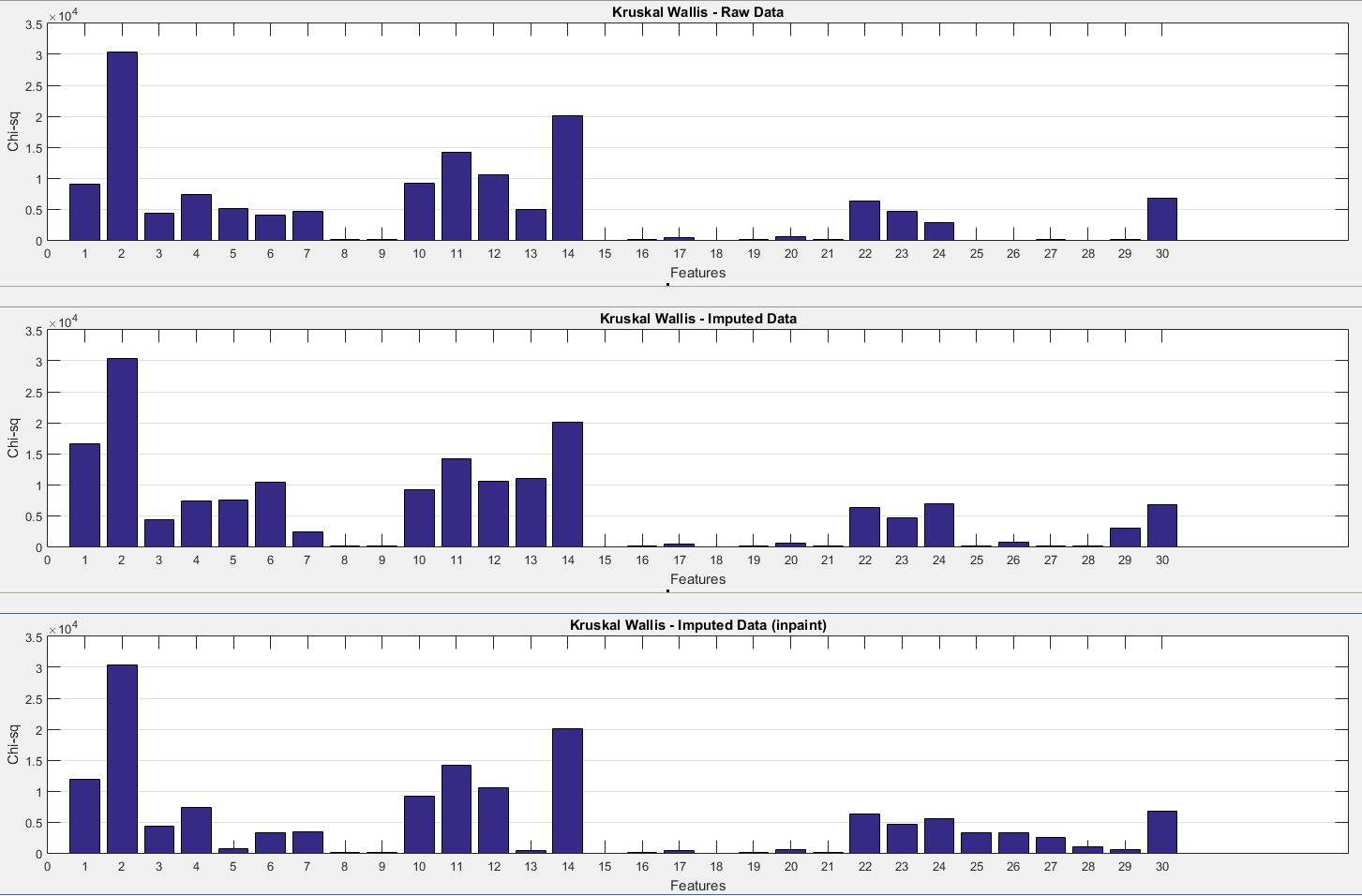


Figure 3Kruskal-Wallis test of the three data sets

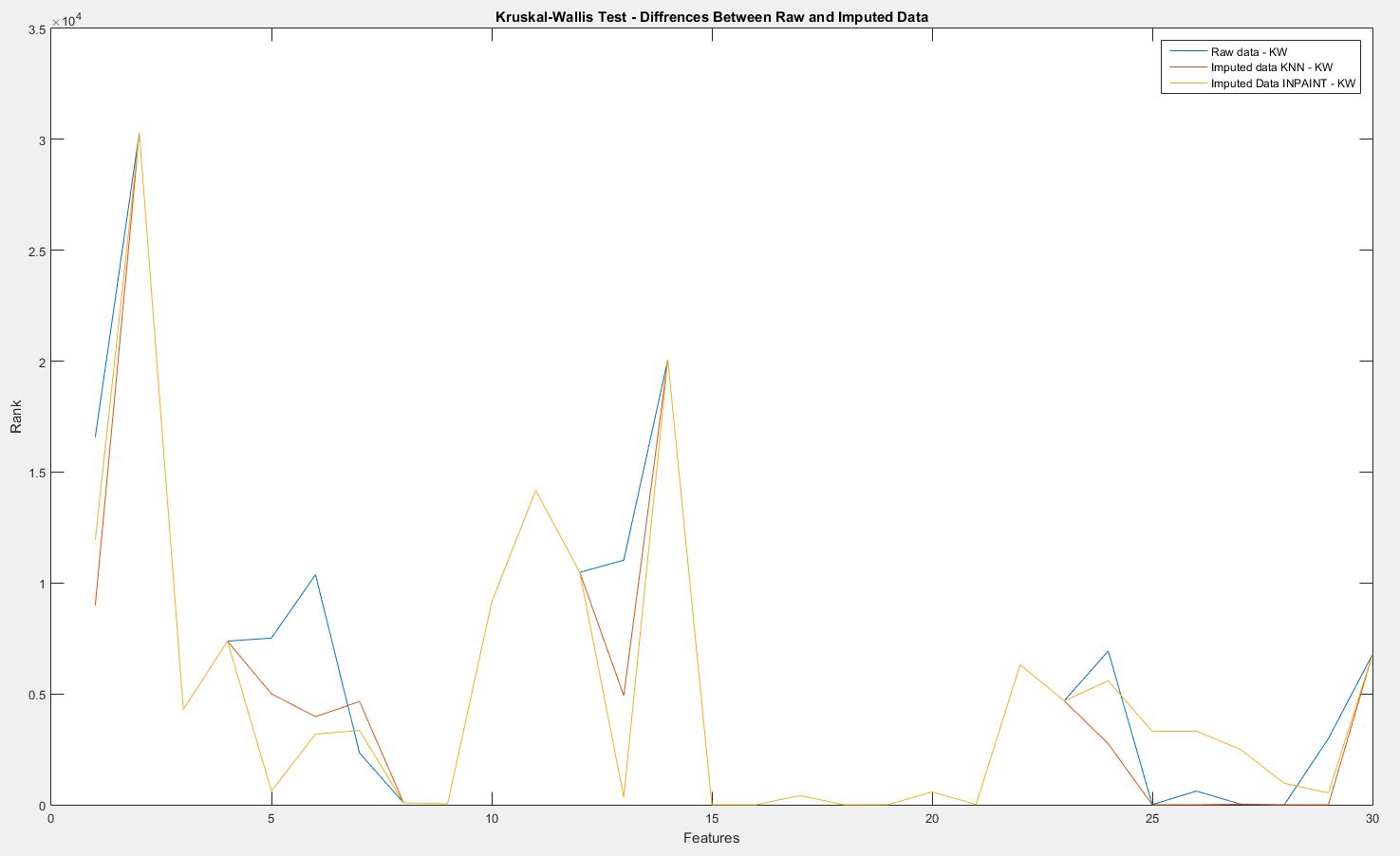


Figure 4Plotting the results- Shows Difference Between Raw And Imputed Data

It is decided to use ‘knnimpute’ instead of ‘inpain\_nans’, because the results are more similar to raw data.

Now we’ll decide if we should use results from PCA ALS or knnimpute.

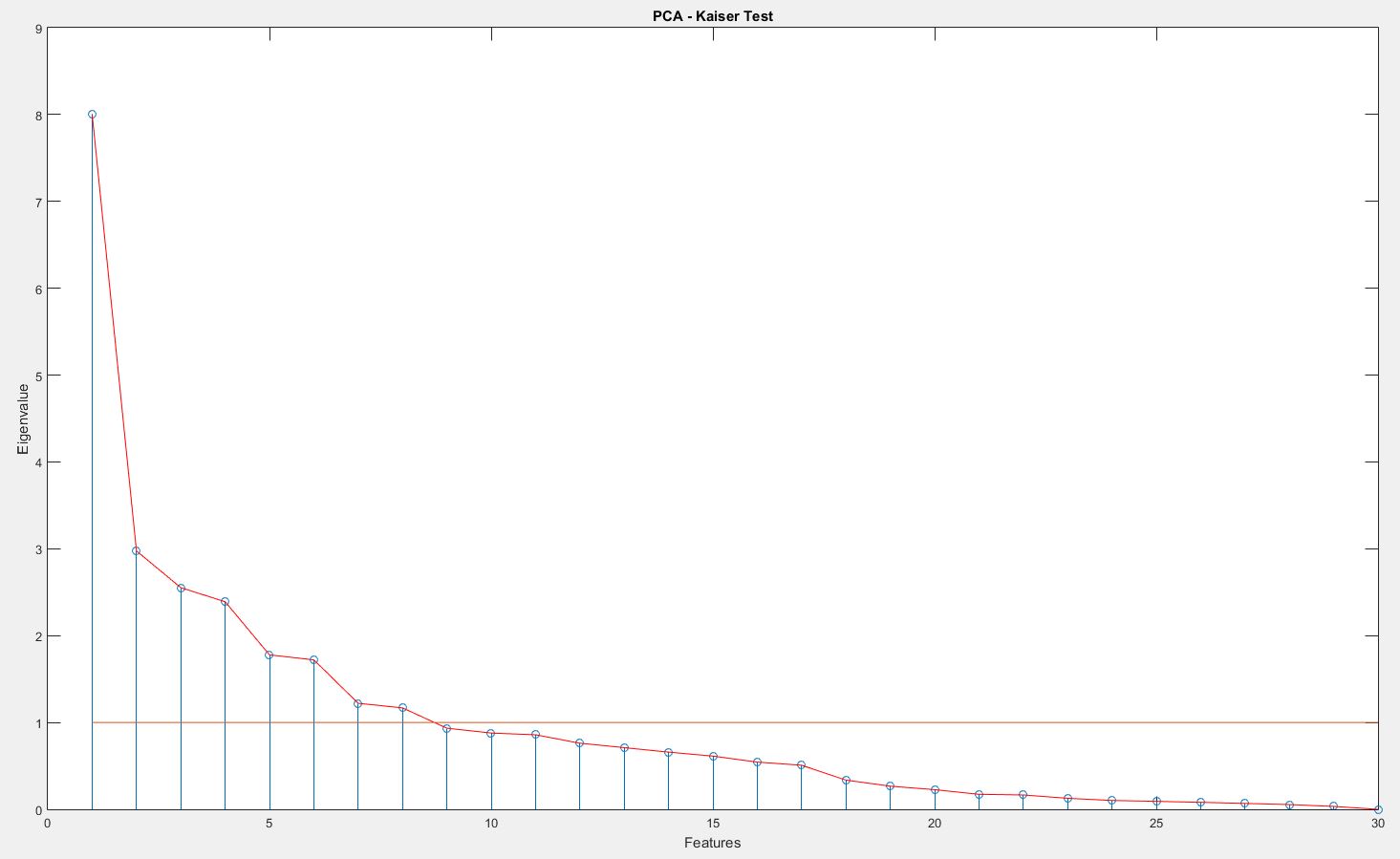


Figure 5 Kaiser test (knnimpute)

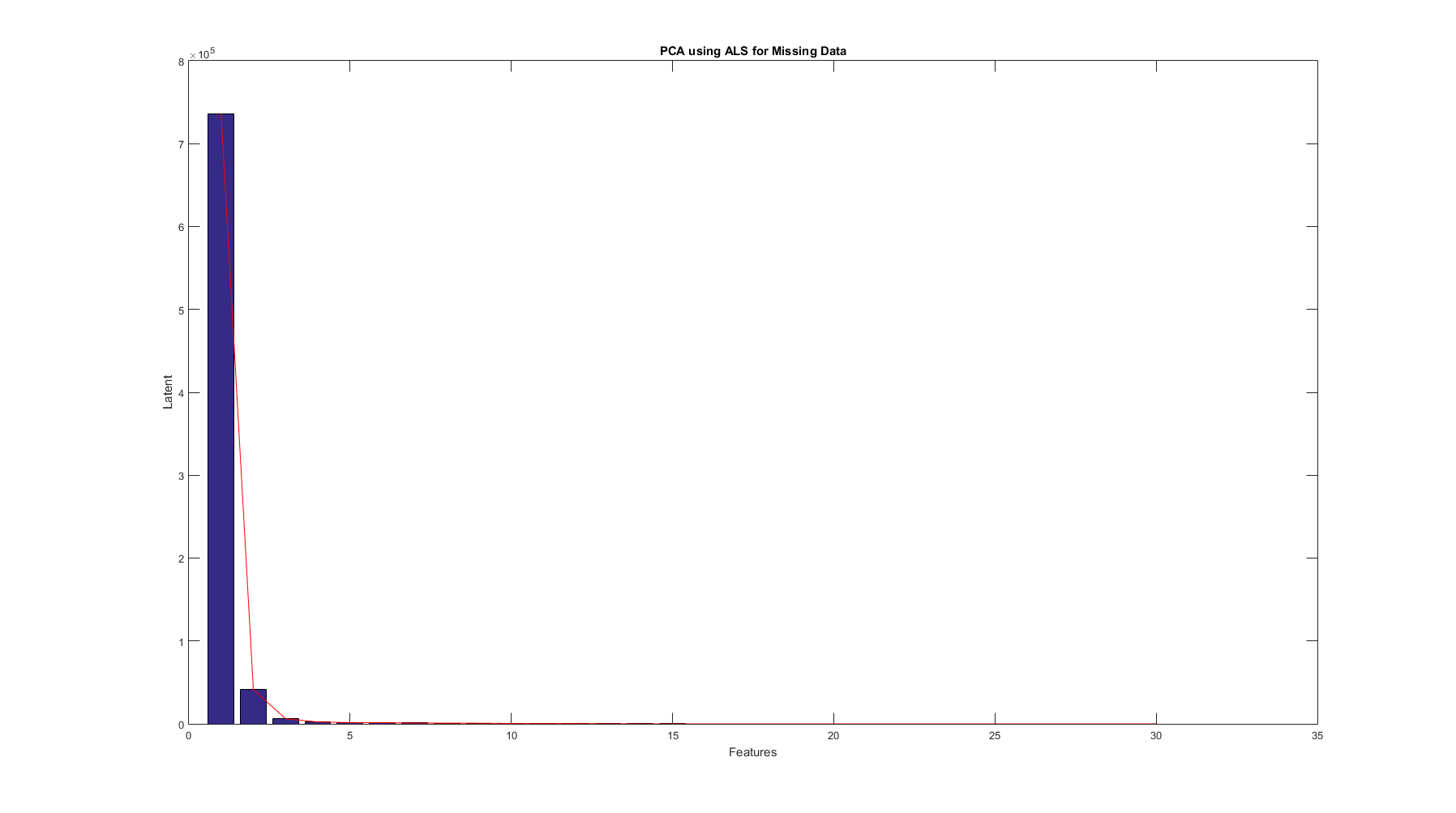


Figure 6 Kaiser test gives only 1feature and Scree – 3

PCA ALS is unacceptable and chose knnimpute.

# **Feature selection and reduction**

To reduce the number of the features we’ll use 4 techniques: Receiver operating characteristic, Kruskal-Wallis, ReliefF algorithm and Sequential feature selection *(+ PCA).*

1. **Receiver operating characteristic (ROC):**

function [AUC] = roc\_rank(data, y, posclass)

AUC = zeros(size(data, 2), 1);

for i = 1:size(data, 2)

[~,~,~,AUCcurr] = perfcurve(y, data(:, i), posclass);

AUC(i,1) = AUCcurr;

end

end

We use area under the curve (AUC) in order to calculate the rank of each feature.

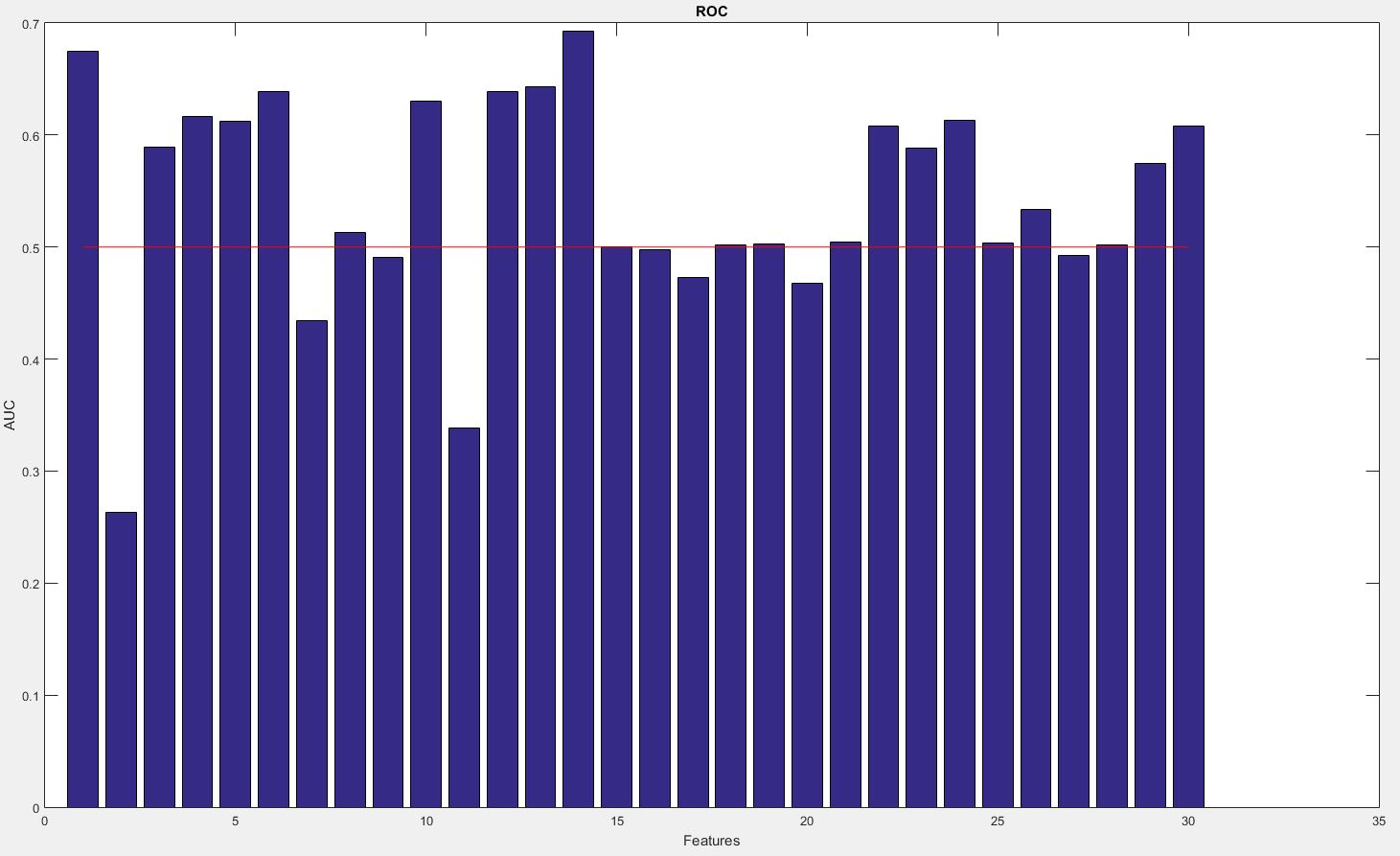


Figure 7ROC

We can use the features with AUC > 0.5, but we decided to classify the features with AUC > 0.47 as acceptable, because several features are very close to 0.5.

The features 1, 3, 4, 5, 6, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30 are selected.

1. **Kruskal-Wallis**

function [chisq] = kw\_rank(data, y)

chisq = zeros(size(data, 2), 1);

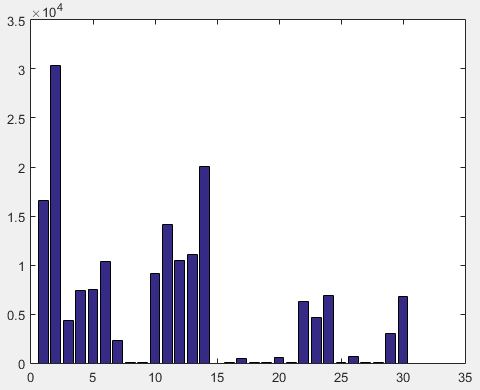
for i = 1:size(data, 2)

[~, table, ~] = kruskalwallis(data(:, i), y, 'off');

chisq(i, 1) = cell2mat( table(2, 5) );

end

end



We decided to “say” that: if (chi-sq(i) > 4700) => feature ‘i’ is acceptable.

The features 1, 2, 4, 5, 6, 10, 11, 12, 13, 14, 22, 24, 30 are selected.

1. **Sequential feature selection (SFS)**

It is used 4 types of discriminant functions: linear, mahalanobis, quadratic, diagonal quadratic for classification. They are executed in sequential forward and backward selection (8 tests) with cross-validation with 10 folds.

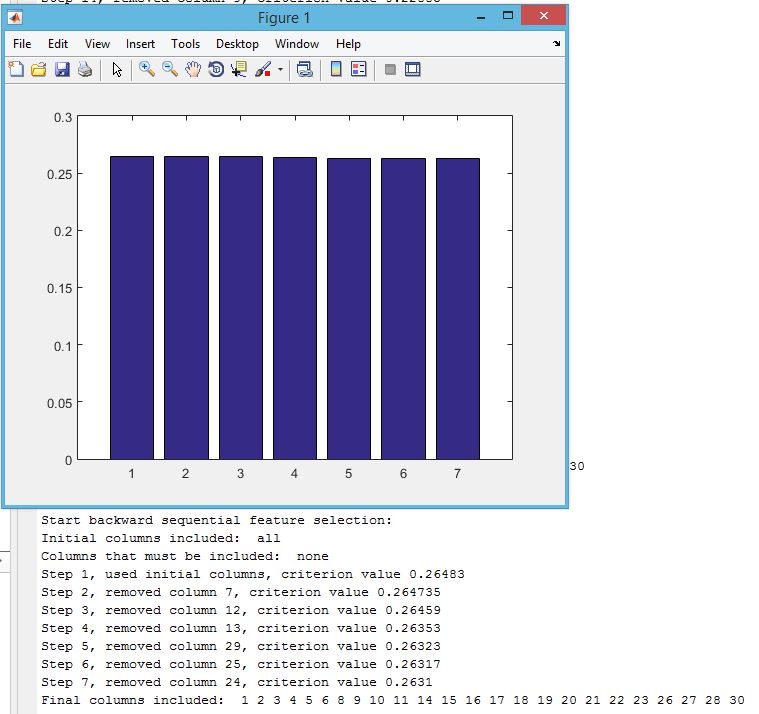


Figure 8 SBS – Linear

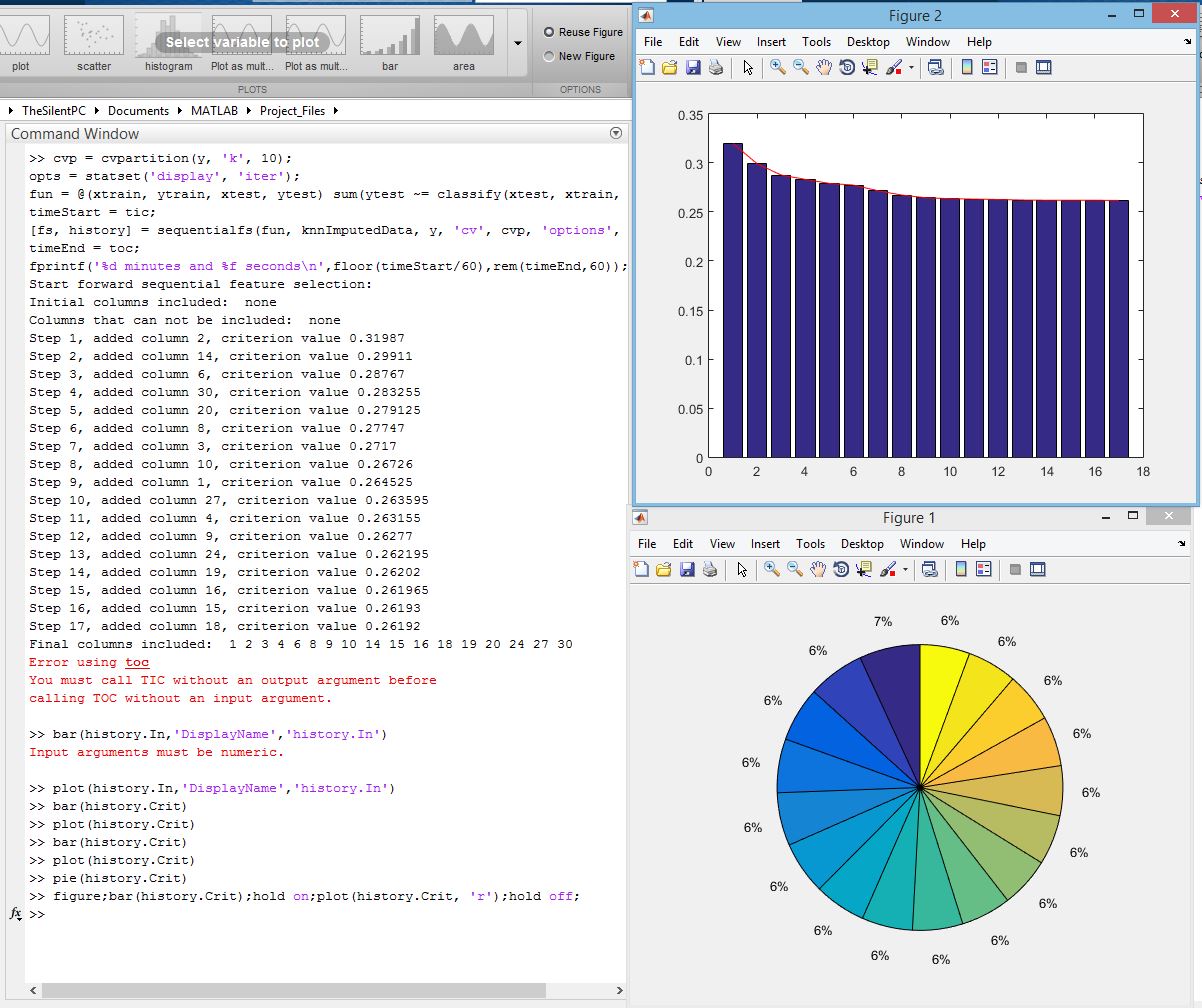


Figure 9SFS – Linear

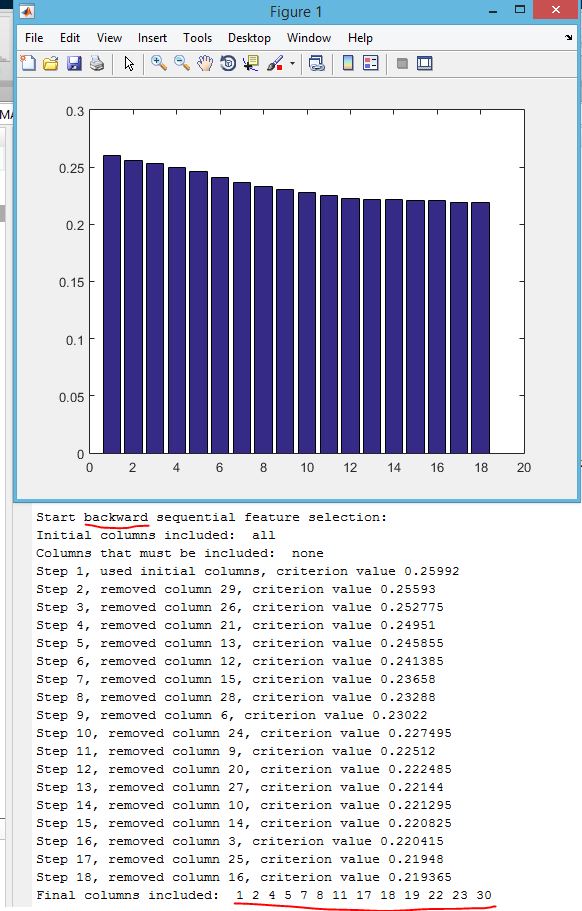


Figure 10 SBS – Mahalanobis

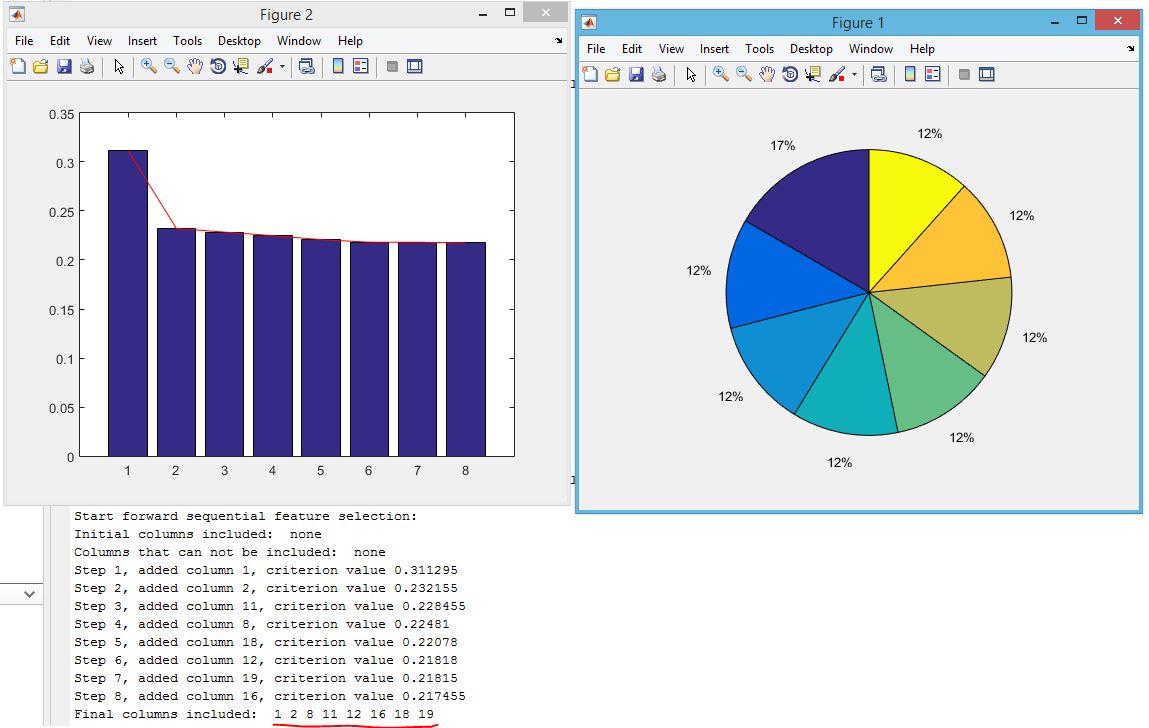


Figure 11 SFS – Mahalonobis

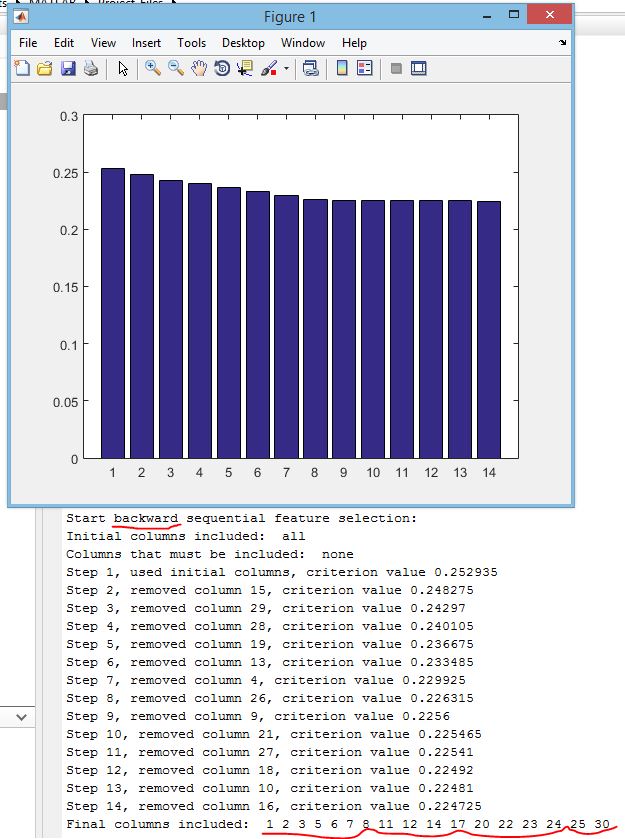


Figure 12SBS – Quadratic

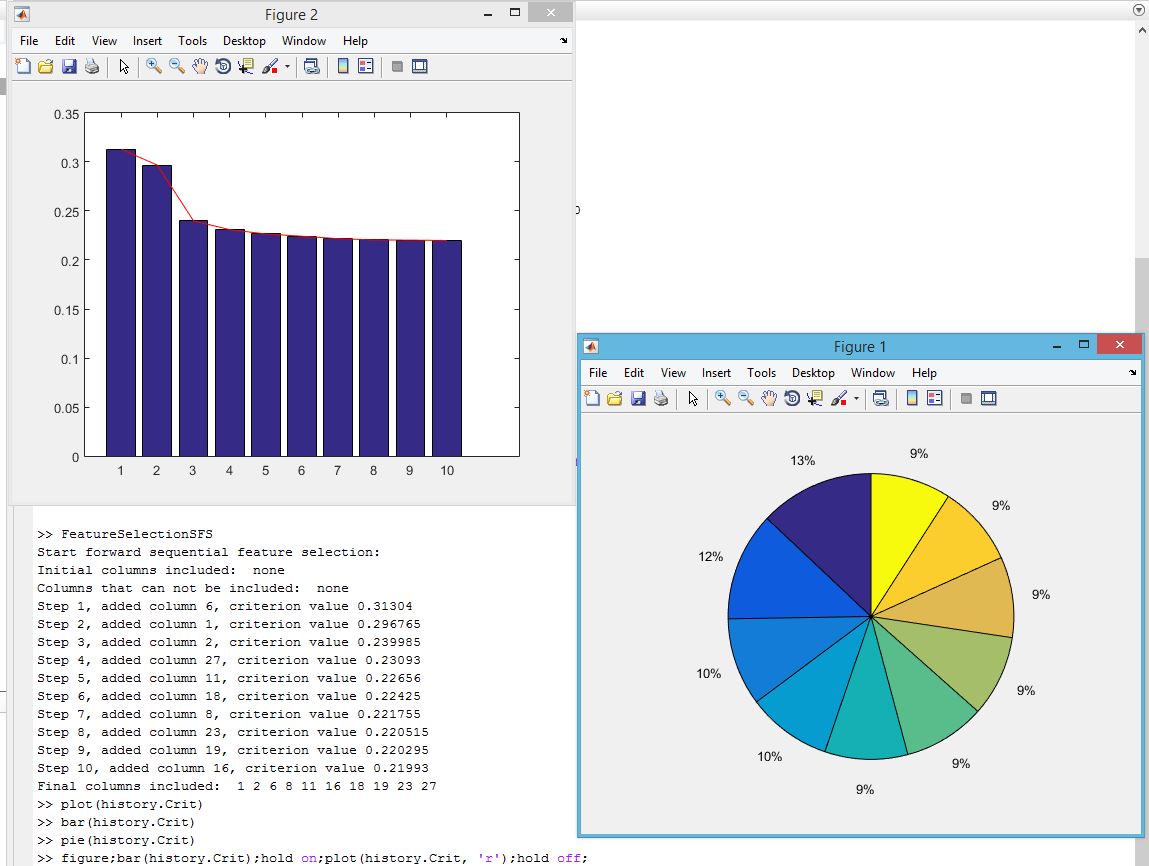


Figure 13SFS – Quadratic

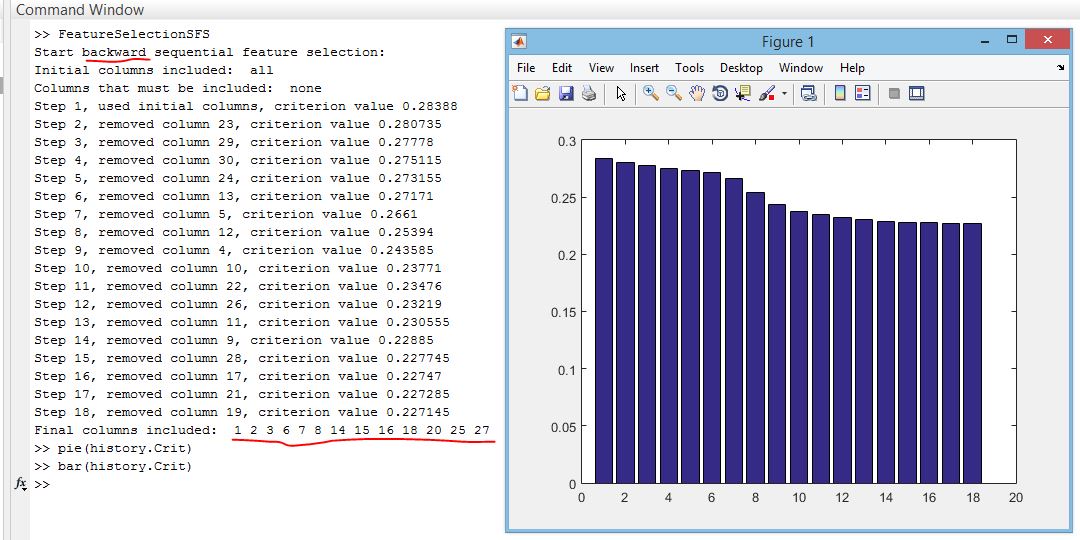


Figure 14SBS – Diagquadratic

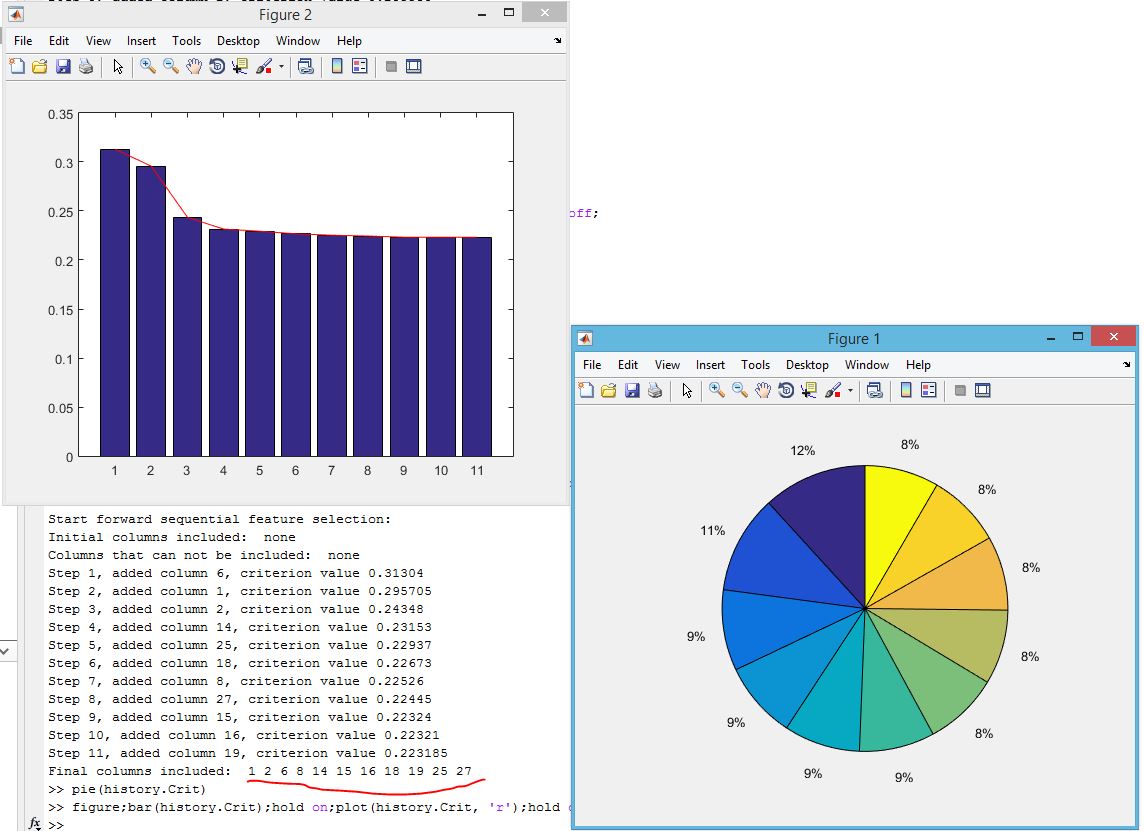


Figure 15 SFS – Diagquadratic

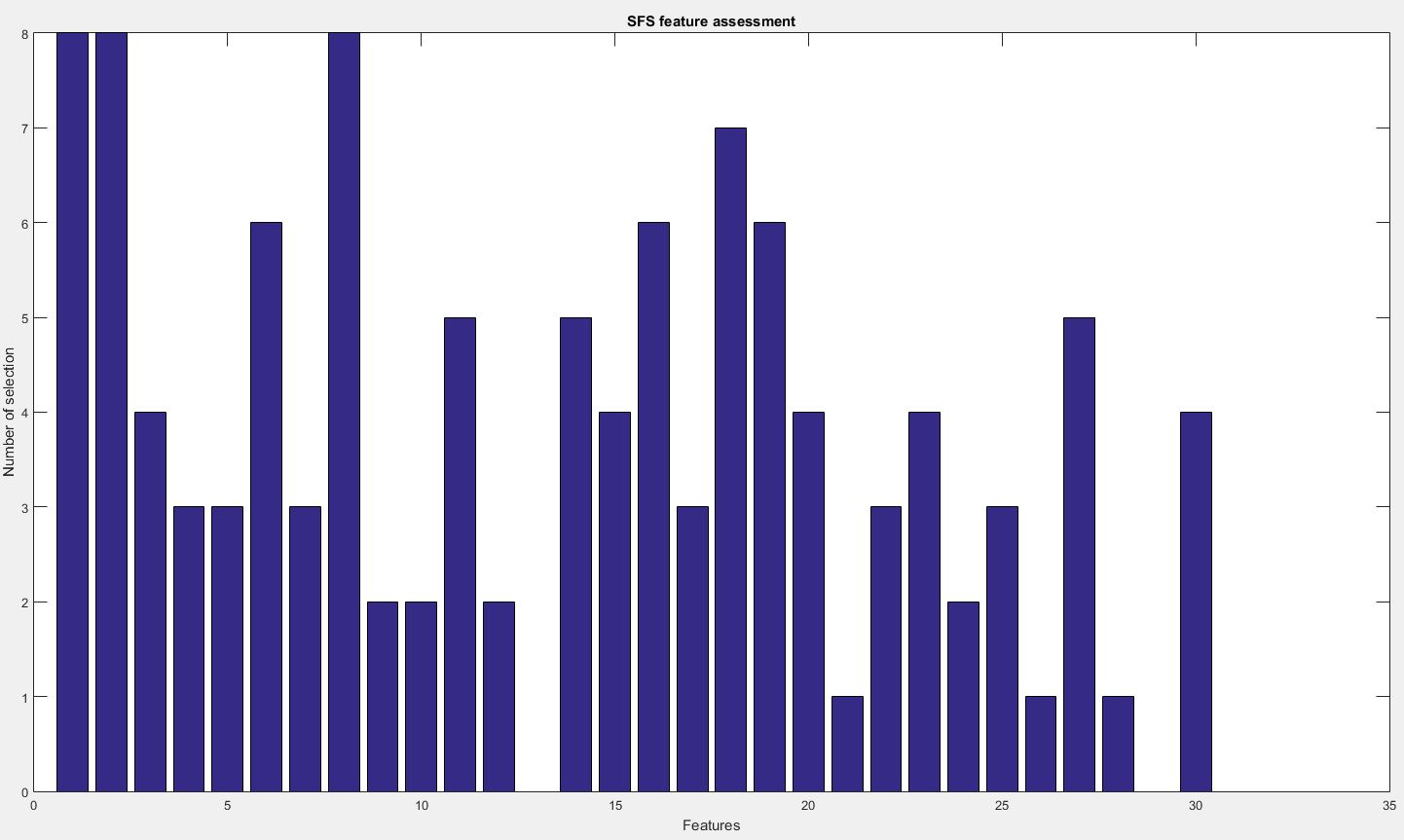


Figure 16Shows which feature how many times is selected

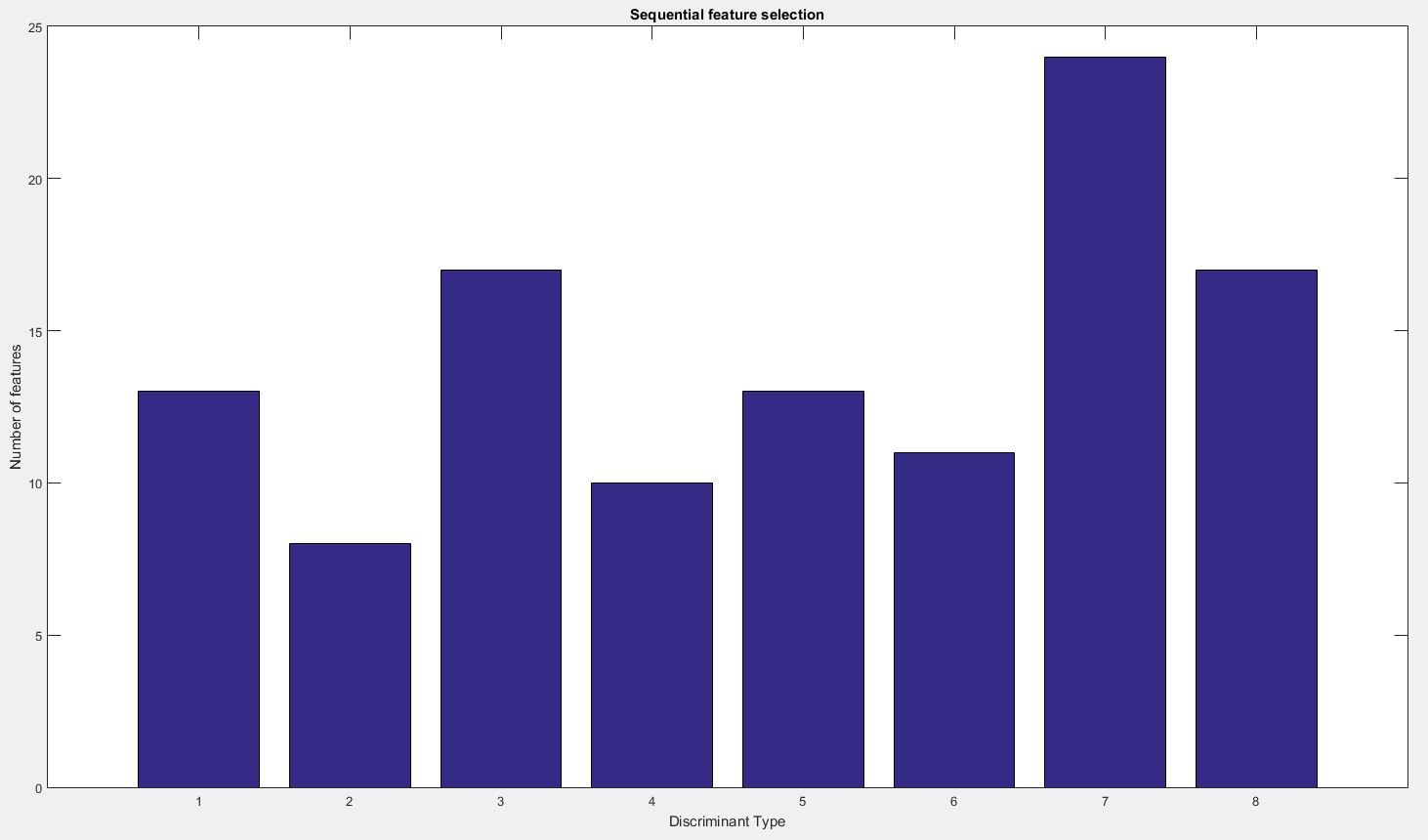


Figure 17 Shows how many features are selected in each test

Discriminant types: **1, 2** – SBS mahalanobis, SFS mahalanobis; **3, 4** - SBS quadratic, SFS, quadratic; **5, 6** – SBS diagqadratic, SFS diagquadratic; **7, 8** – SBS linear, SFS linear.

1. **Importance of attributes (predictors) using ReliefF algorithm**

[rank, weight] = relieff(knnImputedData, y, 10);

% multiplies by 1000 in order to show selected prdictors

figure; bar(weight(rank) \* 1000);

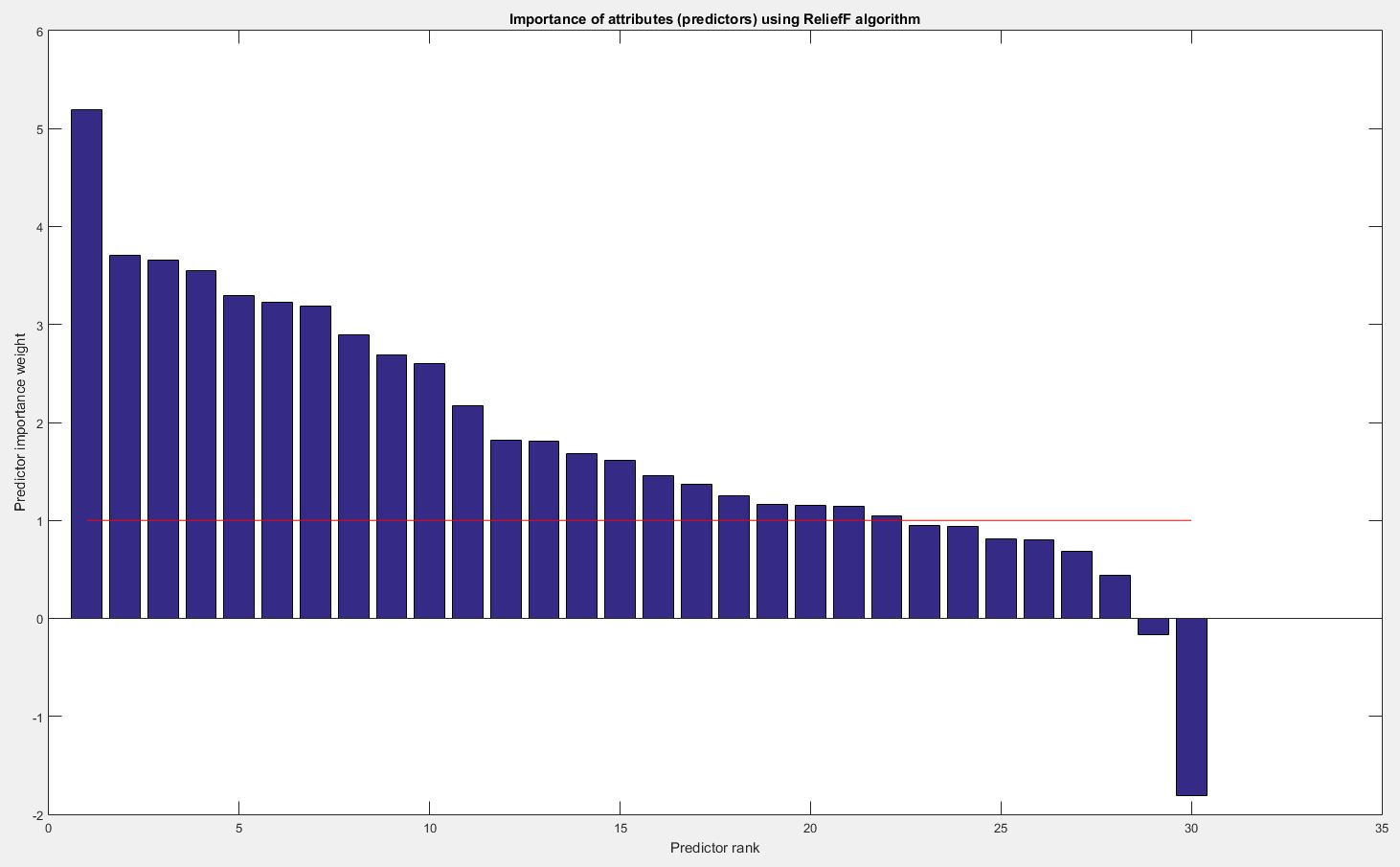
hold on; plot(ones(1,30), 'r');

title('Importance of attributes (predictors) using ReliefF algorithm');

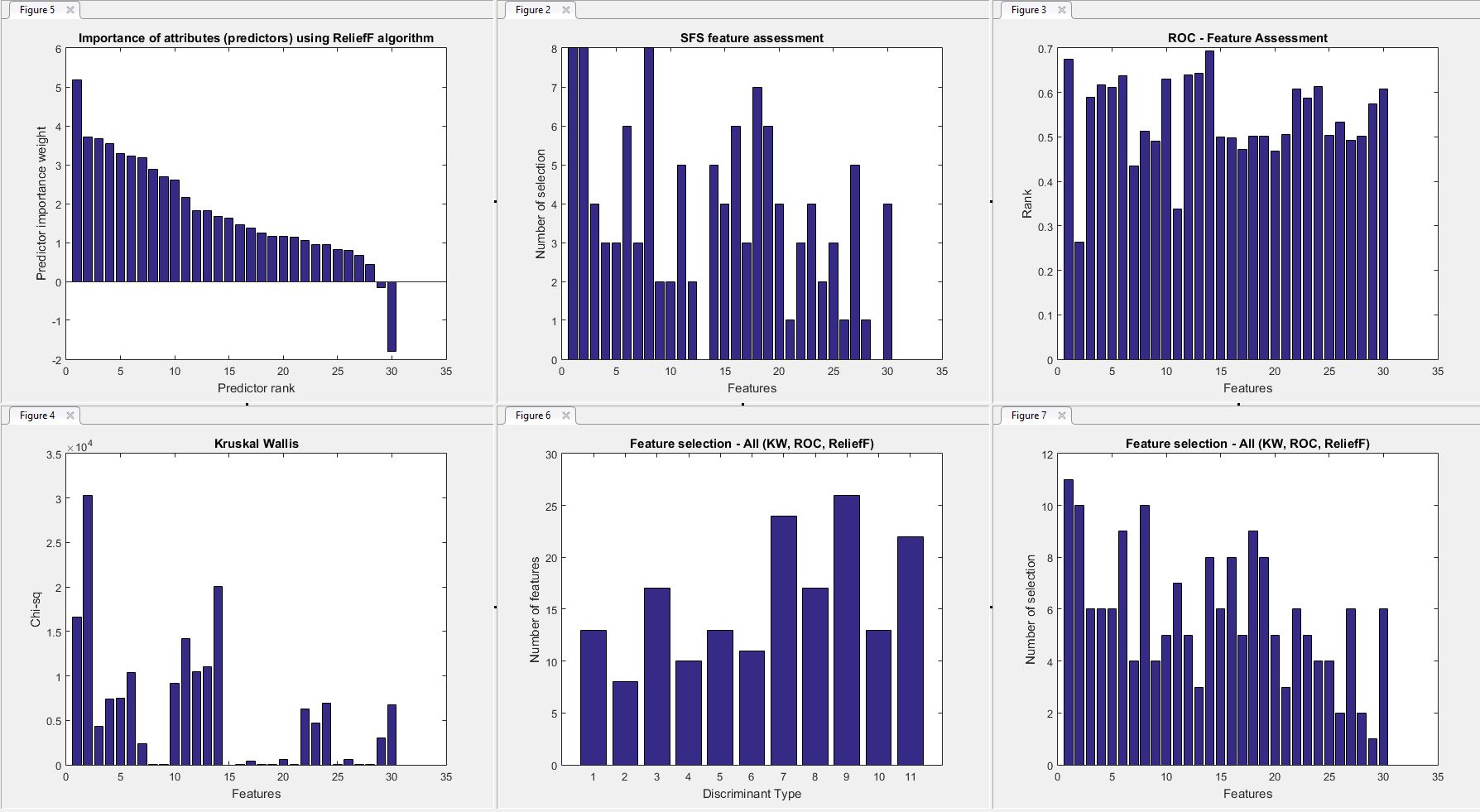
xlabel('Predictor rank');

ylabel('Predictor importance weight');

hold off;



Фигура 1Importance of attributes (predictors) using ReliefF algorithm

In the figures (Figure 6, Figure 7) in positions 2, 2 and 2, 3 it is united the results from **KW, ReliefF, ROC and SFS**.

% calculating how many features to select

selectedFeatures(1:size(summAll, 2)) = ones(); % selected features

featuresNumberForSel = ceil(rms(summ1All)); % root mean square and ceiling

sortedSummAll = sort(summAll, 'descend'); % sort the data in order to get

max = sortedSummAll(1, featuresNumberForSel); % value of the 'featuresNumberForSel'th feature

% if there are another features with value=max, they will be selected also.

for i=1:size(summAll, 2)

if (summAll(1,i) < max)

selectedFeatures(1, i) = 0;

end

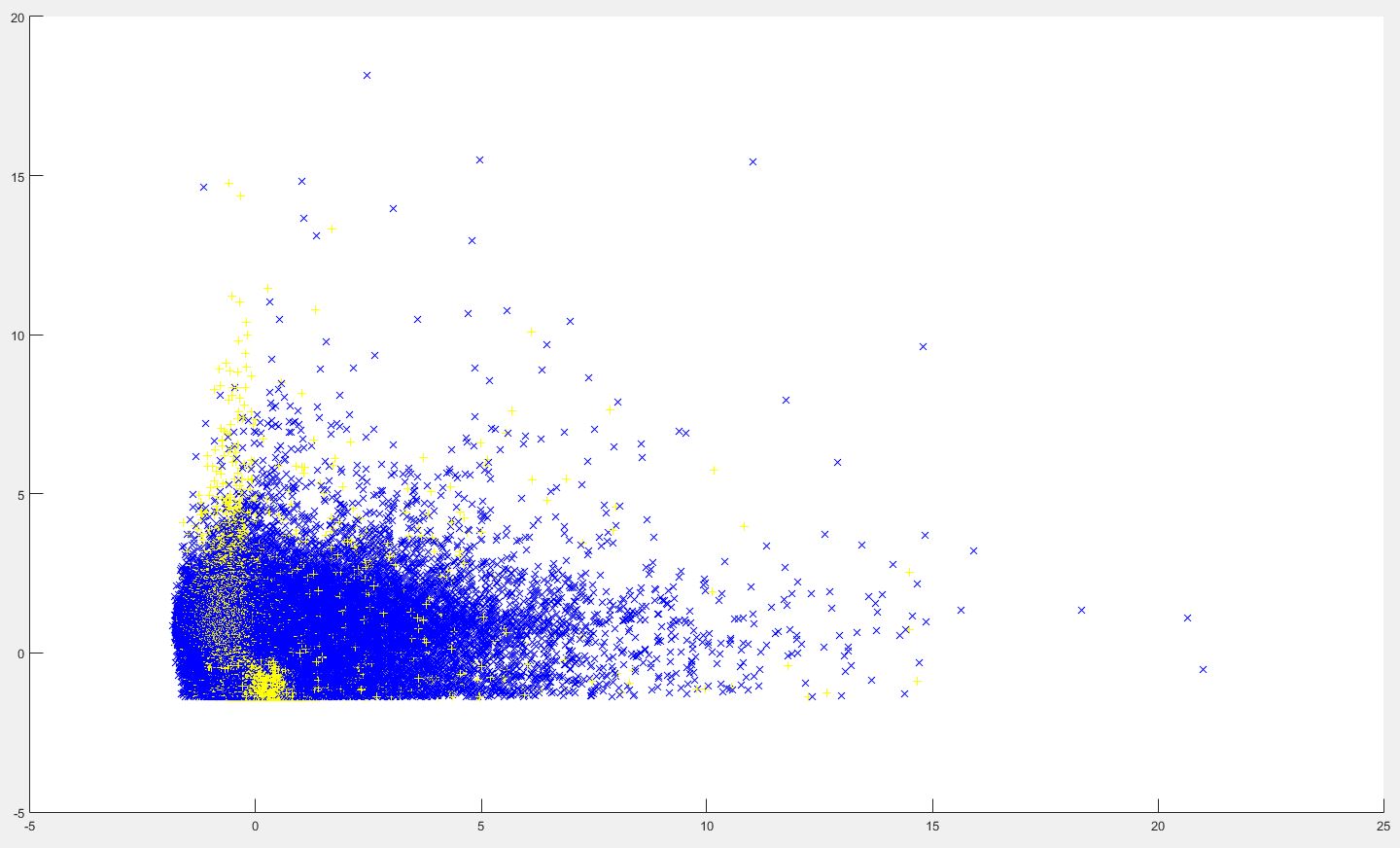
end

selectedFeaturesNumber = sum(selectedFeatures);

We already have the selected features, now we should remove redundant features:

reduced\_data = reduce\_dimention(knnImputedData, selectedFeatures);

figure; ppatterns(reduced\_data', y');



It is made cross validation of the data before and after reducing and the results are very close:

obj = fitcdiscr(**reduced\_data**,y); cvm = crossval(obj); L = kfoldLoss(cvm) **=> 0.2539**  
obj = fitcdiscr(**knnImputedData**,y); cvm = crossval(obj); L = kfoldLoss(cvm) **=> 0.2531**

classf = @(XTRAIN, ytrain,XTEST)(classify(XTEST,XTRAIN,ytrain));

cvMCR = crossval('mcr',**reduced\_data**,y,'predfun',classf) **=> 0.2655**

cvMCR = crossval('mcr',**knnImputedData**,y,'predfun',classf) **=> 0.2648**

opts = statset('UseParallel',true);

regf=@(XTRAIN,ytrain,XTEST)(XTEST\*regress(ytrain,XTRAIN));

cvMse = crossval('mse',**knnImputedData**,y,'Predfun',regf,'Options',opts) **=> 0.6014**

cvMse = crossval('mse',**reduced\_data**,y,'Predfun',regf,'Options',opts) **=> 0.6015**