Table of Contents


```
% DEMO_MUSE
%
% This script compares MUSE and mRMR feature selection on a toy dataset.
% It generates synthetic data with informative, redundant, and noise
% features, runs both feature selectors, and evaluates classification
% performance using cross-validated accuracy and AUC.
%
% Author: Sai Sanjay Balaji

clearvars
close all

demo_muse

function demo_muse()

% DEMO_MUSE_VS_MRMR
% Compare MUSE vs mRMR feature selection on a toy dataset using
% cross-validated classification (Accuracy + AUC).
%
% Requirements:
%   - MUSE.m on your MATLAB path
%   - Statistics and Machine Learning Toolbox:
%       fscmrmr, fitcsvm, fitPosterior, perfcurve, cvpartition
%
% What this script does:
%   1) Generates a toy dataset with:
%         - Dinf informative features
%         - Dred redundant (correlated) features derived from informative ones
%         - optional nonlinear informative feature
%         - Dnoise pure noise features
%   2) Runs MUSE to pick up to m features
%   3) Runs mRMR (fscmrmr) to rank features and takes the top m
%   4) For k = 1, .., k_Max, trains a classifier (linear SVM) on the top-k
features
%      and evaluates performance with K-fold CV.
%   5) Plots Accuracy and AUC vs k for both methods.
```

Toy Data Settings

```
rng(7);           % controls  randomness
N = 8000;         % samples
Dinf = 12;        % informative features (any positive integer)
Dred = 20;        % redundant features (>=0)
Dnoise = 20;      % noise features (>=0)
nonlinear = true; % add a nonlinear informative feature

[X, y] = makeToyData(N, Dinf, Dred, Dnoise, nonlinear);
y = y(:); % ensure column

fprintf('Toy dataset created: N=%d, D=%d, class balance: P(y=0)=%.3f,
P(y=1)=%.3f\n', ...
    size(X,1), size(X,2), mean(y==0), mean(y==1));
```

Feature Selection Parameters

Choose parameters that are reasonable for continuous biomedical features.

```
m  = 20;    % select up to m features
K  = 20;    % number of equiprobable bins for discretization
p  = 0.20;  % fraction of samples used in uncertainty score
T  = 0.10;  % impurity threshold for elimination
Ts = 0.01;  % stop if either class has <Ts fraction of initial survivors (set
0 to disable)
```

Run MUSE

```
[selected_muse, Jhist, elimFrac] = MUSE(X, y, m, K, p, T, Ts);
fprintf('MUSE selected %d features.\n', numel(selected_muse));
```

```
MUSE selected 20 features.
```

Run mRMR

fscmrmr returns a full ranking of features

```
idx_mrmr = fscmrmr(X, categorical(y));
selected_mrmr = idx_mrmr(1:min(m, numel(idx_mrmr)));
fprintf('mRMR selected %d features.\n', numel(selected_mrmr));
```

```
mRMR selected 20 features.
```

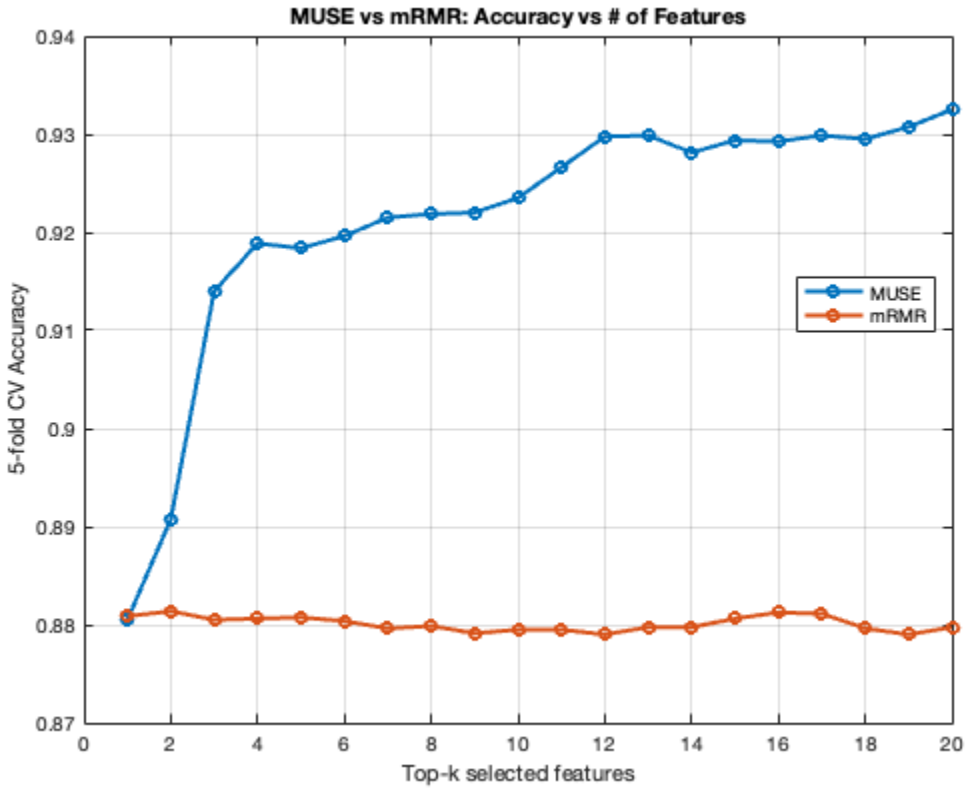
Classification Evaluation

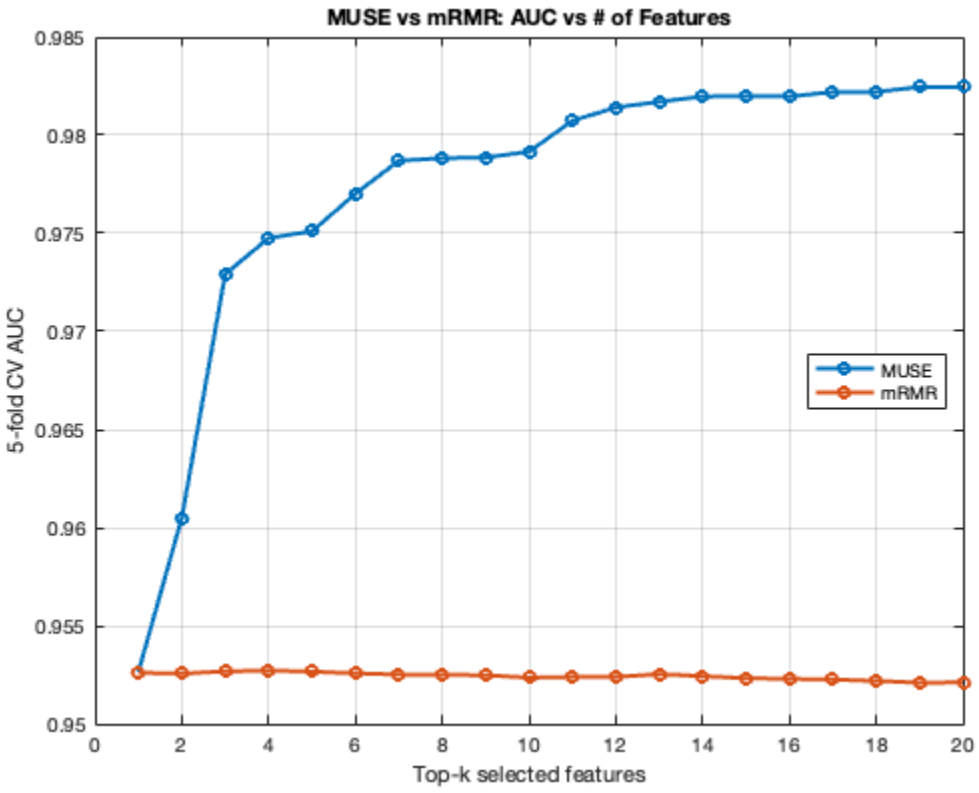
```
kMax = min([m, numel(selected_muse), numel(selected_mrmr)]);
ks = 1:kMax;

modelType = "svm_linear";
Kfold = 5;
```

```
[acc_muse, auc_muse] = evalFS(X, y, selected_muse, ks, modelType, Kfold);
[acc_mrmr, auc_mrmr] = evalFS(X, y, selected_mrmr, ks, modelType, Kfold);
```

Plot Results

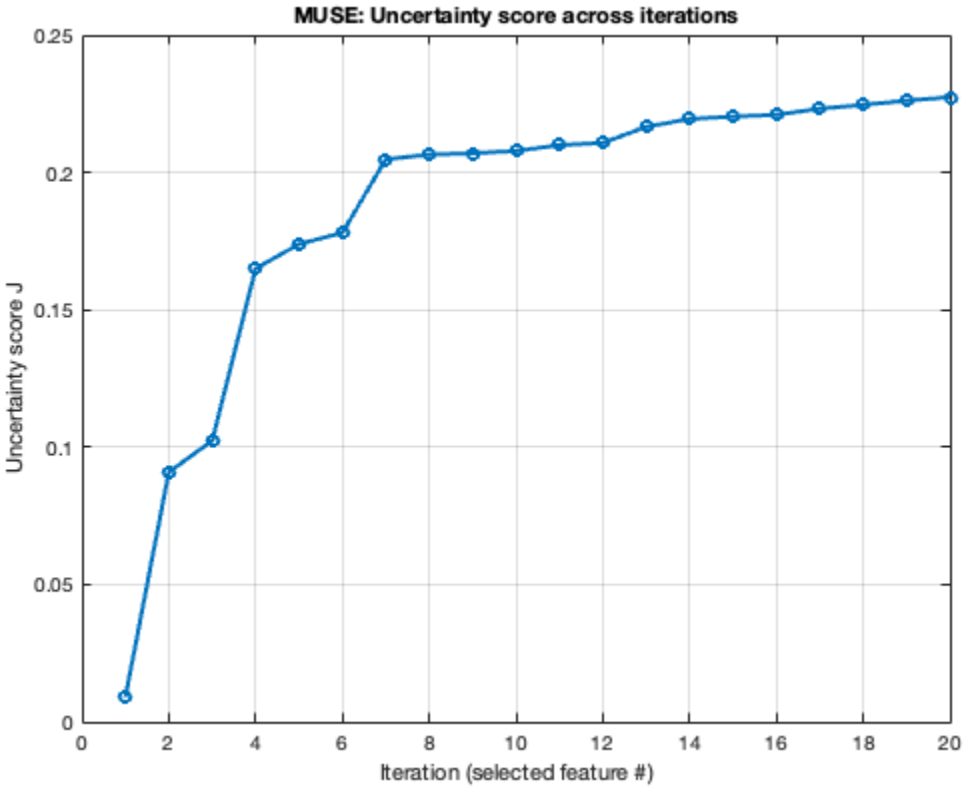
```
figure;
plot(ks, acc_muse, '-o', 'LineWidth', 2); hold on;
plot(ks, acc_mrmr, '-o', 'LineWidth', 2);
grid on;
xlabel('Top-k selected features');
ylabel(sprintf('%d-fold CV Accuracy', Kfold));
title('MUSE vs mRMR: Accuracy vs # of Features');
legend('MUSE', 'mRMR', 'Location', 'best');

figure;
plot(ks, auc_muse, '-o', 'LineWidth', 2); hold on;
plot(ks, auc_mrmr, '-o', 'LineWidth', 2);
grid on;
xlabel('Top-k selected features');
ylabel(sprintf('%d-fold CV AUC', Kfold));
title('MUSE vs mRMR: AUC vs # of Features');
legend('MUSE', 'mRMR', 'Location', 'best');
```

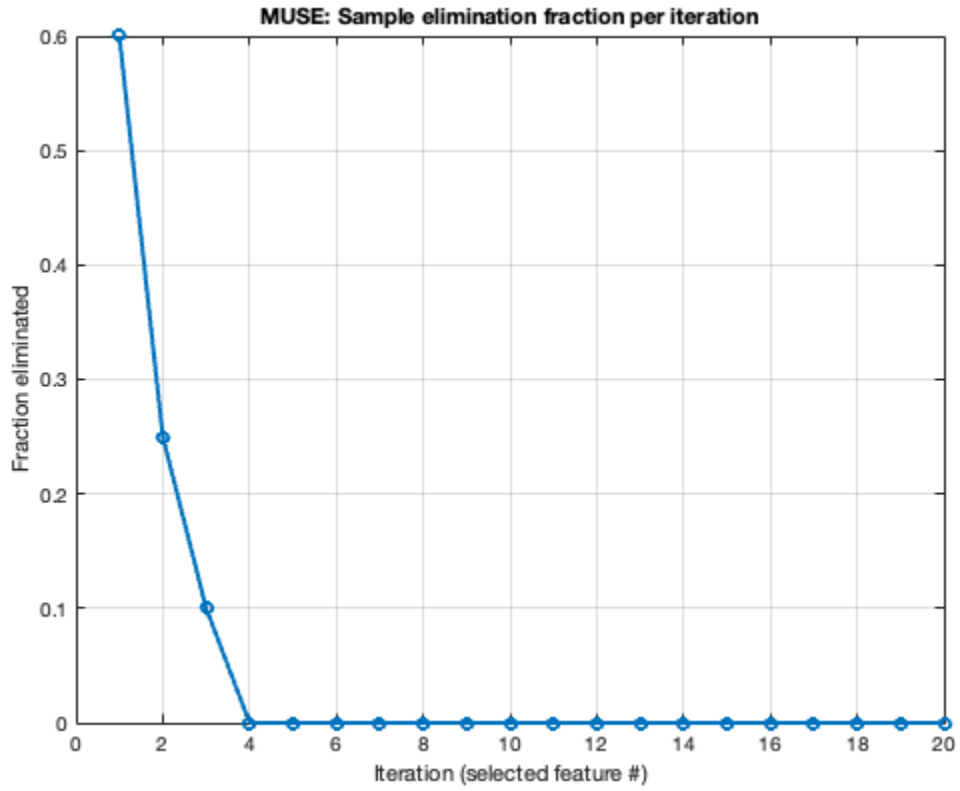




Optional: MUSE dynamics

```
figure;
plot(1:numel(Jhist), Jhist, '-o', 'LineWidth', 2);
grid on;
xlabel('Iteration (selected feature #)');
ylabel('Uncertainty score J');
title('MUSE: Uncertainty score across iterations');

figure;
plot(1:numel(elimFrac), elimFrac, '-o', 'LineWidth', 2);
grid on;
xlabel('Iteration (selected feature #)');
ylabel('Fraction eliminated');
title('MUSE: Sample elimination fraction per iteration');
```


MUSE: Uncertainty score across iterations
Uncertainty score J
Iteration (selected feature #)
0
0.05
0.1
0.15
0.2
0.25
0
2
4
6
8
10
12
14
16
18
20


MUSE: Sample elimination fraction per iteration
Fraction eliminated
Iteration (selected feature #)
0
0.1
0.2
0.3
0.4
0.5
0.6
0
2
4
6
8
10
12
14
16
18
20

```
end
```

Function definition: Toy Dataset Generation

```
function [X, y] = makeToyData(N, Dinf, Dred, Dnoise, nonlinear)
% MAKETOYDATA
% Toy data generator for feature selection tests.
% Works for any Dinf>=1, Dred>=0, Dnoise>=0.
%
% Output:
%   X: [N x D] standardized (z-scored) feature matrix
%   y: [N x 1] binary labels (0/1)

% ---- Labels (balanced as much as possible) ----
y = zeros(N,1);
y(1:floor(N/2)) = 0;
y(floor(N/2)+1:end) = 1;
y = y(randperm(N));

% ---- Informative features ----
% Class 0: mean vector all zeros
mu0 = zeros(1, Dinf);

% Class 1: smoothly decaying separation across informative dimensions
% (no static, hard-coded list)
sep_max = 2.0;   % separation on first informative dimension
sep_min = 0.3;   % separation on last informative dimension
if Dinf == 1
    sep = sep_max;
else
    sep = linspace(sep_max, sep_min, Dinf);
end
mu1 = sep;

% Mild correlation across informative dims (more realistic than identity)
rho = 0.25;
Sigma = (1-rho)*eye(Dinf) + rho*ones(Dinf);

Xinf = zeros(N, Dinf);
n0 = sum(y==0);
n1 = sum(y==1);
Xinf(y==0,:) = mvnrnd(mu0, Sigma, n0);
Xinf(y==1,:) = mvnrnd(mu1, Sigma, n1);

% ---- Redundant features ----
% Create redundant features as noisy linear combinations of informative ones.
Xred = [];
if Dred > 0
    % Fixed random mixing given rng() set in the caller for reproducibility
    W = randn(Dinf, Dred);
    W = W ./ vecnorm(W, 2, 1); % normalize columns

    noise_std = 0.20; % redundancy noise level
```

```
    Xred = Xinf * W + noise_std*randn(N, Dred);
end

% ---- Optional nonlinear informative feature ----
Xnl = [];
if nonlinear
    % Uses first two informative dims if present; else first dim only.
    if Dinf >= 2
        x1 = Xinf(:,1);
        x2 = Xinf(:,2);
        Xnl = tanh(x1 .* x2) + 0.10*randn(N,1);
    else
        x1 = Xinf(:,1);
        Xnl = tanh(x1) + 0.10*randn(N,1);
    end
end

% ---- Noise features ----
Xnoise = [];
if Dnoise > 0
    Xnoise = randn(N, Dnoise);
end

% ---- Combine and standardize ----
X = [Xinf, Xred, Xnl, Xnoise];
X = zscore(X);

end
```

Toy dataset created: N=8000, D=53, class balance: P(y=0)=0.500, P(y=1)=0.500

Function definition: Evaluate classification performance

```
function [acc, auc] = evalFS(X, y, selected, ks, modelType, Kfold)
% EVALFS
% Evaluate classification performance using top-k features (k in ks)
% from the ranking/selection `selected`.
%
% Outputs:
%   acc: mean CV accuracy for each k
%   auc: mean CV AUC for each k

selected = selected(:)';
kMax = max(ks);
selected = selected(1:min(kMax, numel(selected)));

cvp = cvpartition(y, 'KFold', Kfold);

acc = nan(size(ks));
auc = nan(size(ks));
```

```
for ii = 1:numel(ks)
    k = ks(ii);
    feat = selected(1:k);

    foldAcc = nan(Kfold,1);
    foldAuc = nan(Kfold,1);

    for f = 1:Kfold
        tr = training(cvp, f);
        te = test(cvp, f);

        Xtr = X(tr, feat);
        ytr = y(tr);
        Xte = X(te, feat);
        yte = y(te);

        switch modelType
            case "svm_linear" % Only linear SVM is shown here for the demo
                mdl = fitcsvm(Xtr, ytr, ...
                    'KernelFunction', 'linear', ...
                    'Standardize', false, ...
                    'ClassNames', [0 1]);

                % Calibrate to get posterior probabilities for AUC
                mdl = fitPosterior(mdl, Xtr, ytr);

                [pred, score] = predict(mdl, Xte);
                foldAcc(f) = mean(pred == yte);

                % score(:,2) corresponds to P(class=1 | x)
                [~,~,~,AUC] = perfcurve(yte, score(:,2), 1);
                foldAuc(f) = AUC;

            otherwise
                error('Unknown modelType: %s', modelType);
        end
    end

    acc(ii) = mean(foldAcc, 'omitnan');
    auc(ii) = mean(foldAuc, 'omitnan');
end

end
```

Published with MATLAB® R2024b