Supplementary File (1): Hybrid Feature Selection Methods Code - MatLab

This File contains the codes for Hybrid (mRMR+PSO) and Hybrid (LASSO+GA) feature selection methods.

1- Hybrid (mRMR+PSO) Code:

```
% ============================================================
% Hybrid mRMR + Particle Swarm Optimization Feature Selection
% Dataset: mfma.xlsx
%
% Main purposes of this script:
% 1. Read the dataset from Excel.
% 2. Automatically detect the outcome variable as the last column.
% 3. Exclude the outcome from the predictors.
% 4. Optionally exclude screening/diagnostic variables for sensitivity analysis.
% 5. Convert predictors into numeric format.
% 6. Use mRMR as the first-stage feature-ranking method.
% 7. Automatically choose the number of mRMR candidate features using CV-AUC.
% 8. Use binary PSO as the second-stage optimization method.
% 9. Use cross-validated AUC as the PSO fitness criterion.
% 10. Save selected features, mRMR scores, PSO settings, and optimization details.
% 11. Produce a readable ranked bar chart of selected features.
%
% Important methodological note:
% This script avoids an arbitrary fixed feature number.
% mRMR first ranks the candidate predictors and automatically determines
% the candidate subset size using cross-validated AUC.
% PSO then searches for the best subset among the mRMR-retained features.
%
```

```
% In the final nested-CV modeling pipeline, mRMR + PSO should be repeated
% inside each training fold to avoid information leakage.
% ============================================================

%% ============================================================
%  1. Load dataset
% ============================================================

dataPath = 'C:\Users\moham\OneDrive - Arab American University\PhD thesis\Top predictors\Submissions\BMC Medical Informatics and Decision Making\Peer Review 1\mfma.xlsx';

T = readtable(dataPath);

fprintf('\nDataset loaded successfully.\n');
fprintf('Rows: %d\n', height(T));
fprintf('Columns: %d\n', width(T));

%% ============================================================
%  2. Detect outcome variable
% ============================================================

% The last column is assumed to be the outcome variable.
% This outcome column will not be used as a predictor.
targetVar = T.Properties.VariableNames{end};

fprintf('\nOutcome variable detected as: %s\n', targetVar);

% Define positive class.
% Keep this as '1' if CRC cases are coded as 1.
```

```
positiveClass = '1';

%% ============================================================
%  3. Optional exclusion of screening/diagnostic variables
% ============================================================

% For the full-variable analysis, keep this empty.
excludeVars = {};

% For the sensitivity analysis requested by reviewers, add the exact
% variable names for colonoscopy and occult blood testing here.
%
% Example:
% excludeVars = {'didColonoscopy','didOccultBloodTest'};
%
% Important:
% Screening/diagnostic variables should be removed before mRMR and PSO,
% not after feature selection.

%% ============================================================
%  4. Prepare outcome and predictors
% ============================================================

Ycat = categorical(T.(targetVar));

fprintf('\nOutcome categories detected:\n');
disp(categories(Ycat));

classNames = categories(Ycat);
```

```
if ~any(strcmp(classNames, positiveClass))
    error('positiveClass was not found. Check outcome categories and edit positiveClass.');
end

% Remove outcome and optional excluded variables from predictors.
allVars = T.Properties.VariableNames;
predictorVars = setdiff(allVars, [{targetVar}, excludeVars], 'stable');

Xtable = T(:, predictorVars);

fprintf('\nNumber of predictors before cleaning: %d\n', width(Xtable));

%% ============================================================
%  5. Convert predictors to numeric matrix
% ============================================================

% mRMR and PSO-based model evaluation require consistent numeric predictors.
% Text/categorical variables are converted into numeric category codes.
% Numeric predictors are retained as numeric values.

Xnumeric = zeros(height(Xtable), width(Xtable));

for i = 1:width(Xtable)

    xi = Xtable{:, i};

    if iscellstr(xi) || isstring(xi) || ischar(xi)
        xi = categorical(xi);
```

```
    end

    if iscategorical(xi)
        Xnumeric(:, i) = double(xi);
    else
        Xnumeric(:, i) = double(xi);
    end
end

%% ============================================================
%  6. Remove rows with missing outcome
% ============================================================

validRows = ~ismissing(Ycat);
Xtable = Xtable(validRows, :);
Xnumeric = Xnumeric(validRows, :);
Ycat = Ycat(validRows);

%% ============================================================
%  7. Remove invalid predictors
% ============================================================

% Remove predictors that are completely missing or have only one unique value.
% Such variables cannot contribute to classification.

varsToRemove = false(1, width(Xtable));

for i = 1:size(Xnumeric, 2)
```

```
    xi = Xnumeric(:, i);

    if all(isnan(xi))
        varsToRemove(i) = true;
    else
        xiNonMissing = xi(~isnan(xi));
        if numel(unique(xiNonMissing)) <= 1
            varsToRemove(i) = true;
        end
    end
end

removedVars = Xtable.Properties.VariableNames(varsToRemove);

Xtable(:, varsToRemove) = [];
Xnumeric(:, varsToRemove) = [];

fprintf('\nParticipants included: %d\n', height(Xtable));
fprintf('Candidate predictors after cleaning: %d\n', width(Xtable));

if ~isempty(removedVars)
    fprintf('\nRemoved variables because they were empty or had only one unique value:\n');
    disp(removedVars');
end

featureNames = Xtable.Properties.VariableNames;

%% ============================================================
%  8. Handle missing predictor values
```

```
% ============================================================

% Missing values are imputed using the median of each variable.
% In the final nested-CV pipeline, imputation should be fitted inside
% the training folds only.

for i = 1:size(Xnumeric, 2)

   xi = Xnumeric(:, i);

   if any(isnan(xi))
      medianValue = median(xi, 'omitnan');
      xi(isnan(xi)) = medianValue;
      Xnumeric(:, i) = xi;
   end
end

%% ============================================================
%  9. Standardize predictors
% ============================================================

% Standardization is used for stable model fitting and distance-sensitive
% behavior during subset evaluation.
%
% In the final nested-CV pipeline, mean and standard deviation should be
% estimated using training folds only.

mu = mean(Xnumeric, 1, 'omitnan');
sigma = std(Xnumeric, 0, 1, 'omitnan');
```

```
sigma(sigma == 0) = 1;

Xz = (Xnumeric - mu) ./ sigma;

%% ============================================================
%  10. First stage: mRMR ranking and automatic candidate cut-off
% ============================================================

% mRMR ranks features according to maximum relevance with the outcome and
% minimum redundancy among predictors.
%
% The number of mRMR-retained features is not fixed arbitrarily.
% Candidate subset sizes are evaluated using 5-fold CV-AUC, and the best
% subset size is retained for the PSO stage.

rng(2026);

p = size(Xz, 2);

candidateM = unique([5 10 15 20 25 30 p]);
candidateM(candidateM > p) = [];

mrmrCVFolds = 5;
cvpMrmr = cvpartition(Ycat, 'KFold', mrmrCVFolds);

meanAUC_mrmr = zeros(numel(candidateM), 1);
sdAUC_mrmr = zeros(numel(candidateM), 1);
```

```
fprintf('\nSelecting automatic mRMR candidate cut-off...\n');

for mIdx = 1:numel(candidateM)

    m = candidateM(mIdx);
    aucPerFold = nan(mrmrCVFolds, 1);

    for fold = 1:mrmrCVFolds

        trainIdx = training(cvpMrmr, fold);
        testIdx = test(cvpMrmr, fold);

        Xtrain = Xz(trainIdx, :);
        Ytrain = Ycat(trainIdx);

        Xtest = Xz(testIdx, :);
        Ytest = Ycat(testIdx);

        % Leakage-prevention step:
        % mRMR ranking is performed on the training fold only.
        [rankedIdxFold, ~] = fscmrmr(Xtrain, Ytrain);

        selectedIdxFold = rankedIdxFold(1:m);

        % Ensemble classifier is used only to choose the mRMR candidate size.
        mdl = fitcensemble(Xtrain(:, selectedIdxFold), Ytrain, ...
            'Method', 'Bag', ...
            'NumLearningCycles', 100);
```

```
        [~, scores] = predict(mdl, Xtest(:, selectedIdxFold));

        modelClassNames = cellstr(mdl.ClassNames);
        posCol = find(strcmp(modelClassNames, positiveClass));

        if isempty(posCol)
            error('Positive class not found in model class names.');
        end

        [~, ~, ~, aucValue] = perfcurve(Ytest, scores(:, posCol), categorical({positiveClass}));

        aucPerFold(fold) = aucValue;
    end

    meanAUC_mrmr(mIdx) = mean(aucPerFold, 'omitnan');
    sdAUC_mrmr(mIdx) = std(aucPerFold, 'omitnan');

    fprintf('Top %d mRMR candidates: Mean AUC = %.4f, SD = %.4f\n', ...
        m, meanAUC_mrmr(mIdx), sdAUC_mrmr(mIdx));
end

% Select mRMR candidate size with highest mean AUC.
% If tied, choose the smaller subset.
bestMrmrAUC = max(meanAUC_mrmr);
bestMrmrCandidates = candidateM(meanAUC_mrmr == bestMrmrAUC);
bestM = min(bestMrmrCandidates);

fprintf('\nOptimal number of mRMR candidate features before PSO: %d\n', bestM);
fprintf('Best mRMR candidate-selection mean CV-AUC: %.4f\n', bestMrmrAUC);
```

```
% Final mRMR ranking on full data for descriptive reporting and PSO input.
% In nested-CV modeling, this must be repeated inside each training fold.
[finalMrmrRankedIdx, finalMrmrScores] = fscmrmr(Xz, Ycat);

mrmrCandidateIdx = finalMrmrRankedIdx(1:bestM);
mrmrCandidateNames = featureNames(mrmrCandidateIdx);
mrmrCandidateScores = finalMrmrScores(mrmrCandidateIdx);

X_pso = Xz(:, mrmrCandidateIdx);

fprintf('\nNumber of predictors retained after mRMR stage: %d\n', numel(mrmrCandidateIdx));

%% ============================================================
%  11. PSO settings
% ============================================================

% This script implements binary PSO manually.
% Each particle position is a binary vector:
% 1 = feature selected
% 0 = feature not selected
%
% The PSO fitness function is:
% mean CV-AUC - penalty for subset size
%
% Because PSO is a maximization procedure here, the algorithm searches for
% the feature subset with the highest penalized CV-AUC.

psoRandomSeed = 2026;
```

```
rng(psoRandomSeed);

nParticles = 50;
maxIterations = 100;

inertiaWeight = 0.72;
inertiaDamping = 0.99;

cognitiveCoefficient = 1.49;
socialCoefficient = 1.49;

velocityMin = -6;
velocityMax = 6;

psoCVFolds = 5;
complexityPenalty = 0.01;

nPSOFeatures = size(X_pso, 2);

psoSettingsTable = table( ...
   {'Particle Swarm Optimization'; ...
    'Binary particle position'; ...
    'Number of particles'; ...
    'Maximum iterations'; ...
    'Inertia weight'; ...
    'Inertia damping'; ...
    'Cognitive coefficient'; ...
    'Social coefficient'; ...
    'Velocity minimum'; ...
```

```
    'Velocity maximum'; ...
    'Transfer function'; ...
    'Random seed'; ...
    'Fitness criterion'; ...
    'Cross-validation folds in PSO fitness'; ...
    'Complexity penalty'}, ...
   {'Custom binary PSO'; ...
    '1 = selected, 0 = not selected'; ...
    num2str(nParticles); ...
    num2str(maxIterations); ...
    num2str(inertiaWeight); ...
    num2str(inertiaDamping); ...
    num2str(cognitiveCoefficient); ...
    num2str(socialCoefficient); ...
    num2str(velocityMin); ...
    num2str(velocityMax); ...
    'Sigmoid transfer function'; ...
    num2str(psoRandomSeed); ...
    'Mean CV-AUC minus penalty for subset size'; ...
    num2str(psoCVFolds); ...
    ['AUC - ' num2str(complexityPenalty) ' * selected_features / mRMR_candidates']}, ...
   'VariableNames', {'PSO_Setting', 'Value'});

fprintf('\nPSO settings:\n');
disp(psoSettingsTable);

mrmrSettingsTable = table( ...
   {'mRMR function'; ...
    'Candidate cut-off rule'; ...
```

```
    'Candidate subset sizes tested'; ...
    'Cross-validation folds for cut-off'; ...
    'Cut-off metric'; ...
    'Tie-breaking rule'; ...
    'Selected number of mRMR candidates'}, ...
   {'fscmrmr'; ...
    'Automatic cut-off based on CV-AUC'; ...
    mat2str(candidateM); ...
    num2str(mrmrCVFolds); ...
    'Mean validation AUC'; ...
    'Smaller subset preferred'; ...
    num2str(bestM)}, ...
   'VariableNames', {'mRMR_Setting', 'Value'});

%% ============================================================
%  12. Initialize PSO population
% ============================================================

% Particle positions are initialized randomly as binary vectors.
% Velocities are initialized randomly in the allowed velocity range.

position = rand(nParticles, nPSOFeatures) > 0.5;
velocity = velocityMin + (velocityMax - velocityMin) * rand(nParticles, nPSOFeatures);

% Ensure each particle selects at least one feature.
for i = 1:nParticles
   if ~any(position(i, :))
      randomFeature = randi(nPSOFeatures);
      position(i, randomFeature) = true;
```

```
    end
end

personalBestPosition = position;
personalBestFitness = -inf(nParticles, 1);

globalBestPosition = false(1, nPSOFeatures);
globalBestFitness = -inf;

fitnessHistory = nan(maxIterations, 1);
selectedCountHistory = nan(maxIterations, 1);

%% ============================================================
%  13. Run binary PSO
% ============================================================

fprintf('\nRunning binary PSO feature selection...\n');

for iter = 1:maxIterations

    for i = 1:nParticles

        currentPosition = position(i, :);

        currentFitness = evaluatePSOFitness( ...
            currentPosition, ...
            X_pso, ...
            Ycat, ...
            positiveClass, ...
```

```
        psoCVFolds, ...
        complexityPenalty);

    % Update personal best.
    if currentFitness > personalBestFitness(i)
        personalBestFitness(i) = currentFitness;
        personalBestPosition(i, :) = currentPosition;
    end

    % Update global best.
    if currentFitness > globalBestFitness
        globalBestFitness = currentFitness;
        globalBestPosition = currentPosition;
    end
end

fitnessHistory(iter) = globalBestFitness;
selectedCountHistory(iter) = sum(globalBestPosition);

fprintf('Iteration %d/%d | Best fitness = %.5f | Selected features = %d\n', ...
    iter, maxIterations, globalBestFitness, sum(globalBestPosition));

% Update velocities and positions.
for i = 1:nParticles

    r1 = rand(1, nPSOFeatures);
    r2 = rand(1, nPSOFeatures);

    cognitiveTerm = cognitiveCoefficient .* r1 .* (personalBestPosition(i, :) - position(i, :));
```

```
        socialTerm = socialCoefficient .* r2 .* (globalBestPosition - position(i, :));

        velocity(i, :) = inertiaWeight .* velocity(i, :) + cognitiveTerm + socialTerm;

        velocity(i, :) = max(velocity(i, :), velocityMin);
        velocity(i, :) = min(velocity(i, :), velocityMax);

        probability = 1 ./ (1 + exp(-velocity(i, :)));

        position(i, :) = rand(1, nPSOFeatures) < probability;

        % Ensure each particle selects at least one feature.
        if ~any(position(i, :))
            randomFeature = randi(nPSOFeatures);
            position(i, randomFeature) = true;
        end
    end

    inertiaWeight = inertiaWeight * inertiaDamping;
end

%% ============================================================
%  14. Extract selected features from PSO result
% ============================================================

psoSelectedLocalIdx = find(globalBestPosition == 1);

% Safety fallback:
% If PSO selects no features, select the strongest mRMR candidate.
```

```
if isempty(psoSelectedLocalIdx)
   fprintf('\nPSO selected no features. Applying fallback to strongest mRMR feature.\n');
   [~, strongestIdx] = max(mrmrCandidateScores);
   psoSelectedLocalIdx = strongestIdx;
end

psoSelectedOriginalIdx = mrmrCandidateIdx(psoSelectedLocalIdx);

selectedFeatures = featureNames(psoSelectedOriginalIdx);
selectedMrmrScores = finalMrmrScores(psoSelectedOriginalIdx);

%% ============================================================
%  15. Calculate final hybrid importance scores
% ============================================================

% Hybrid score is based on the mRMR score of PSO-selected features.
% Interpretation:
% mRMR provides the feature ranking and within-method importance score.
% PSO determines whether an mRMR-retained feature is selected in the final subset.

selectedScores = selectedMrmrScores;

% Remove zero or negative scores, if any.
positiveScoreIdx = selectedScores > 0;

selectedFeatures = selectedFeatures(positiveScoreIdx);
selectedScores = selectedScores(positiveScoreIdx);

if isempty(selectedScores)
```

```
    error('All selected mRMR + PSO scores were zero or negative. Please review mRMR and PSO settings.');
end

selectedScoresNorm = selectedScores ./ max(selectedScores);

prettySelectedFeatures = makePrettyLabels(selectedFeatures);

resultsTable = table( ...
    selectedFeatures(:), ...
    prettySelectedFeatures(:), ...
    selectedScores(:), ...
    selectedScoresNorm(:), ...
    'VariableNames', {'Original_Feature_Name', 'Readable_Feature_Label', 'mRMR_Score_after_PSO_Selection', 'Normalized_Hybrid_mRMR_PSO_Score'});

fprintf('\nSelected Hybrid mRMR + PSO features:\n');
disp(resultsTable);

%% ============================================================
%  16. Evaluate final selected subset using CV-AUC for reporting
% ============================================================

[finalMeanAUC, finalSDAUC] = evaluateSubsetAUC( ...
    Xz(:, psoSelectedOriginalIdx), ...
    Ycat, ...
    positiveClass, ...
    psoCVFolds);
```

```
psoSummaryTable = table( ...
   bestM, ...
   numel(selectedFeatures), ...
   globalBestFitness, ...
   finalMeanAUC, ...
   finalSDAUC, ...
   maxIterations, ...
   nParticles, ...
   'VariableNames', {'mRMR_Candidate_Features', 'PSO_Selected_Features', 'Best_PSO_Fitness', 'Final_Selected_Subset_Mean_AUC', 'Final_Selected_Subset_SD_AUC', 'PSO_Iterations', 'PSO_Particles'});

fprintf('\nPSO optimization summary:\n');
disp(psoSummaryTable);

psoHistoryTable = table( ...
   (1:maxIterations)', ...
   fitnessHistory(:), ...
   selectedCountHistory(:), ...
   'VariableNames', {'Iteration', 'Best_Fitness', 'Selected_Feature_Count'});

%% ============================================================
%  17. Save output tables
% ============================================================

outputFolder = fileparts(dataPath);

writetable(resultsTable, fullfile(outputFolder, 'Hybrid_mRMR_PSO_Selected_Features.xlsx'));
writetable(psoSettingsTable, fullfile(outputFolder, 'Hybrid_mRMR_PSO_Settings.xlsx'));
```

```
writetable(mrmrSettingsTable, fullfile(outputFolder, 'Hybrid_mRMR_Settings.xlsx'));
writetable(psoSummaryTable, fullfile(outputFolder, 'Hybrid_mRMR_PSO_Summary.xlsx'));
writetable(psoHistoryTable, fullfile(outputFolder, 'Hybrid_mRMR_PSO_Optimization_History.xlsx'));

mrmrCandidateTable = table( ...
  mrmrCandidateNames(:), ...
  makePrettyLabels(mrmrCandidateNames(:)), ...
  mrmrCandidateScores(:), ...
  mrmrCandidateScores(:) ./ max(mrmrCandidateScores), ...
  'VariableNames', {'Original_Feature_Name', 'Readable_Feature_Label', 'mRMR_Score', 'Normalized_mRMR_Score'});

writetable(mrmrCandidateTable, fullfile(outputFolder, 'Hybrid_mRMR_Candidate_Features.xlsx'));

mrmrCutoffTable = table( ...
  candidateM(:), ...
  meanAUC_mrmr(:), ...
  sdAUC_mrmr(:), ...
  'VariableNames', {'Number_of_mRMR_Candidate_Features', 'Mean_AUC', 'SD_AUC'});

writetable(mrmrCutoffTable, fullfile(outputFolder, 'Hybrid_mRMR_AutoCutoff_Results.xlsx'));

%% ============================================================
%  18. Produce readable bar chart
% ============================================================

[sortedScores, sortIdx] = sort(selectedScoresNorm, 'ascend');
sortedFeatures = selectedFeatures(sortIdx);
sortedLabels = makePrettyLabels(sortedFeatures);
```

```
figure('Color', 'w', 'Position', [100 100 1500 900]);

barh(sortedScores, 0.70);

yticks(1:numel(sortedLabels));
yticklabels(sortedLabels);

ax = gca;
ax.FontSize = 10;
ax.TickLabelInterpreter = 'none';

xlabel('Normalized Hybrid mRMR + PSO Importance Score', 'FontSize', 12);
title('Selected Features Ranked by Hybrid mRMR + PSO', 'FontSize', 14, 'FontWeight', 'bold');

xlim([0, max(sortedScores) * 1.35]);

% Use five decimal places to avoid rounding small values to 0.00.
for i = 1:numel(sortedScores)
   text(sortedScores(i) + 0.015, i, sprintf('%.5f', sortedScores(i)), ...
      'VerticalAlignment', 'middle', ...
      'FontSize', 9);
end

grid on;
box on;

set(gcf, 'PaperPositionMode', 'auto');
```

```
saveas(gcf, fullfile(outputFolder, 'Hybrid_mRMR_PSO_Selected_Features_BarChart.png'));
savefig(gcf, fullfile(outputFolder, 'Hybrid_mRMR_PSO_Selected_Features_BarChart.fig'));

exportgraphics(gcf, fullfile(outputFolder,
'Hybrid_mRMR_PSO_Selected_Features_BarChart_HighResolution.png'), 'Resolution', 300);

fprintf('\nDone.\n');
fprintf('Results saved in:\n%s\n', outputFolder);

%% ============================================================
%  19. Helper function: PSO fitness
% ============================================================

function fitness = evaluatePSOFitness(position, X, Ycat, positiveClass, cvFolds, penaltyWeight)

  % Binary position:
  % 1 = feature selected
  % 0 = feature not selected

  selectedIdx = find(position == 1);

  % If no feature is selected, return very poor fitness.
  if isempty(selectedIdx)
    fitness = -1e6;
    return
  end

  Xsubset = X(:, selectedIdx);
```

```
    try
        [meanAUC, ~] = evaluateSubsetAUC(Xsubset, Ycat, positiveClass, cvFolds);

        subsetPenalty = penaltyWeight * (numel(selectedIdx) / size(X, 2));

        fitness = meanAUC - subsetPenalty;

    catch
        fitness = -1e6;
    end
end

%% ============================================================
%  20. Helper function: evaluate subset using CV-AUC
% ============================================================

function [meanAUC, sdAUC] = evaluateSubsetAUC(Xsubset, Ycat, positiveClass, cvFolds)

    cvp = cvpartition(Ycat, 'KFold', cvFolds);

    aucValues = nan(cvFolds, 1);

    for fold = 1:cvFolds

        trainIdx = training(cvp, fold);
        testIdx = test(cvp, fold);

        Xtrain = Xsubset(trainIdx, :);
        Ytrain = Ycat(trainIdx);
```

```
    Xtest = Xsubset(testIdx, :);
    Ytest = Ycat(testIdx);

    % Ensemble model is used inside PSO fitness to evaluate subsets.
    % This keeps the fitness function consistent with the study focus on
    % ensemble-based predictive performance.
    mdl = fitcensemble(Xtrain, Ytrain, ...
      'Method', 'Bag', ...
      'NumLearningCycles', 100);

    [~, scores] = predict(mdl, Xtest);

    modelClassNames = cellstr(mdl.ClassNames);
    posCol = find(strcmp(modelClassNames, positiveClass));

    if isempty(posCol)
      error('Positive class not found in model class names.');
    end

    [~, ~, ~, aucValue] = perfcurve(Ytest, scores(:, posCol), categorical({positiveClass}));

    aucValues(fold) = aucValue;
  end

  meanAUC = mean(aucValues, 'omitnan');
  sdAUC = std(aucValues, 'omitnan');
end
```

```
%% ============================================================
%  21. Helper function: readable labels
% ============================================================

function prettyLabels = makePrettyLabels(featureNames)

   prettyLabels = cell(size(featureNames));

   for j = 1:numel(featureNames)

      label = char(featureNames{j});

      label = strrep(label, '_', ' ');
      label = strrep(label, '-', ' ');

      label = regexprep(label, '([a-z])([A-Z])', '$1 $2');
      label = regexprep(label, '([A-Z])([A-Z][a-z])', '$1 $2');
      label = regexprep(label, '([0-9])([A-Za-z])', '$1 $2');
      label = regexprep(label, '([A-Za-z])([0-9])', '$1 $2');

      label = strrep(label, ' Of ', ' of ');
      label = strrep(label, ' In ', ' in ');
      label = strrep(label, ' The ', ' the ');
      label = strrep(label, ' And ', ' and ');
      label = strrep(label, ' For ', ' for ');

      label = strrep(label, 'P A', 'Physical Activity');
      label = strrep(label, 'PA', 'Physical Activity');
      label = strrep(label, 'B M I', 'BMI');
```

```
    label = strrep(label, 'C 8', 'C8');
    label = strrep(label, 'C8 Past Smoking', 'Past Smoking');
    label = strrep(label, 'C 8 Past Smoking', 'Past Smoking');

    label = strrep(label, 'Age of Smoking Init iation', 'Age of Smoking Initiation');
    label = strrep(label, 'Age of Smoking Cessation', 'Age of Smoking Cessation');
    label = strrep(label, 'Years Since Smoking', 'Years Since Smoking');
    label = strrep(label, 'Cigars Smoked Weekly', 'Cigars Smoked Weekly');

    label = strrep(label, 'Frequency of Eating Fruits', 'Frequency of Eating Fruits');
    label = strrep(label, 'Frequency of Eating Fruit', 'Frequency of Eating Fruits');
    label = strrep(label, 'Frequency of Eating Vegetables', 'Frequency of Eating Vegetables');
    label = strrep(label, 'Frequency of Eating Red Meat', 'Frequency of Eating Red Meat');
    label = strrep(label, 'Frequency of Eating Grilled Red Meat', 'Frequency of Eating Grilled Red
Meat');
    label = strrep(label, 'Frequency of Eating Chicken', 'Frequency of Eating Chicken');
    label = strrep(label, 'Frequency of Eating Grilled Chicken', 'Frequency of Eating Grilled
Chicken');

    label = strrep(label, 'place of Living', 'Place of Living');
    label = strrep(label, 'Place of Living', 'Place of Living');
    label = strrep(label, 'yearly Income', 'Yearly Income');
    label = strrep(label, 'occupation', 'Occupation');
    label = strrep(label, 'Residence', 'Residence');
    label = strrep(label, 'Marital Status', 'Marital Status');
    label = strrep(label, 'work yes No', 'Work Status');

    label = strrep(label, 'Hand Rolled Cigarettes Smoked Weekly', 'Hand-Rolled Cigarettes Smoked
Weekly');
```

```
    label = strrep(label, 'Manufactured Cigarettes Smoked Weekly', 'Manufactured Cigarettes Smoked Weekly');

    label = strrep(label, 'Pipe Full of Tobacco Smoked Weekly', 'Pipe Full of Tobacco Smoked Weekly');


    label = strrep(label, 'No of Times Consumed Alcohol in the Past 12 Months', 'Alcohol Consumption Frequency in the Past 12 Months');

    label = strrep(label, 'Any Alcohol in the Past 12 Months', 'Any Alcohol Consumption in the Past 12 Months');

    label = strrep(label, 'Occasions of Drinking Alcohol', 'Occasions of Drinking Alcohol');

    label = strrep(label, 'Times of Drinking in the Past 30 Days', 'Times of Drinking in the Past 30 Days');

    label = strrep(label, 'Drinks in the Past 30 Days', 'Drinks in the Past 30 Days');

    label = strrep(label, 'Alcohol Consumption', 'Alcohol Consumption');

    label = strrep(label, 'Quit Alcohol for Health Problem', 'Quit Alcohol for Health Problem');


    label = strrep(label, 'Quit Smoking Last 12 Months', 'Quit Smoking in the Last 12 Months');


    label = strtrim(regexprep(label, '\s+', ' '));


    prettyLabels{j} = label;
  end
end
```

2- Hybrid (LASSO+GA)

```
% ============================================================
% Hybrid LASSO + Genetic Algorithm Feature Selection
% Dataset: mfma.xlsx
%
% Main purposes of this script:
% 1. Read the dataset from Excel.
% 2. Automatically detect the outcome variable as the last column.
% 3. Exclude the outcome from the predictors.
% 4. Optionally exclude screening/diagnostic variables for sensitivity analysis.
% 5. Convert predictors into numeric format.
% 6. Standardize predictors.
% 7. Apply LASSO logistic regression as the first feature-reduction stage.
% 8. Apply Genetic Algorithm as the second optimization stage.
% 9. Use cross-validated AUC as the GA fitness criterion.
% 10. Save selected features, hybrid scores, GA settings, and optimization details.
% 11. Produce a readable ranked bar chart of selected features.
%
% Important methodological note:
% This script avoids an arbitrary fixed feature number.
% LASSO first reduces the feature space using cross-validation.
% GA then automatically searches for the best feature subset among the
% LASSO-retained features.
%
% In the final nested-CV modeling pipeline, LASSO + GA should be repeated
% inside each training fold to avoid information leakage.
% ============================================================
```

```
%% ============================================================
%  1. Load dataset
% ============================================================

dataPath = 'C:\Users\moham\OneDrive - Arab American University\PhD thesis\Top predictors\Submissions\BMC Medical Informatics and Decision Making\Peer Review 1\mfma.xlsx';

T = readtable(dataPath);

fprintf('\nDataset loaded successfully.\n');
fprintf('Rows: %d\n', height(T));
fprintf('Columns: %d\n', width(T));

%% ============================================================
%  2. Detect outcome variable
% ============================================================

% The last column is assumed to be the outcome variable.
% This outcome column will not be used as a predictor.
targetVar = T.Properties.VariableNames{end};

fprintf('\nOutcome variable detected as: %s\n', targetVar);

% Define positive class.
% Keep this as '1' if CRC cases are coded as 1.
positiveClass = '1';

%% ============================================================
```

```
%  3. Optional exclusion of screening/diagnostic variables
% ============================================================

% For the full-variable analysis, keep this empty.
excludeVars = {};

% For the sensitivity analysis requested by reviewers, add the exact
% variable names for colonoscopy and occult blood testing here.
%
% Example:
% excludeVars = {'didColonoscopy','didOccultBloodTest'};
%
% Important:
% Screening/diagnostic variables should be removed before LASSO and GA,
% not after feature selection.

%% ============================================================
%  4. Prepare outcome and predictors
% ============================================================

Ycat = categorical(T.(targetVar));

fprintf('\nOutcome categories detected:\n');
disp(categories(Ycat));

% Check positive class
classNames = categories(Ycat);

if ~any(strcmp(classNames, positiveClass))
```

```
    error('positiveClass was not found. Check outcome categories and edit positiveClass.');
end

% Convert outcome to binary numeric values: positive class = 1, others = 0.
Y = double(Ycat == categorical({positiveClass}));

% Remove outcome and optional excluded variables from predictors.
allVars = T.Properties.VariableNames;
predictorVars = setdiff(allVars, [{targetVar}, excludeVars], 'stable');

Xtable = T(:, predictorVars);

fprintf('\nNumber of predictors before cleaning: %d\n', width(Xtable));

%% ============================================================
%  5. Convert predictors to numeric matrix
% ============================================================

% LASSO and GA-based classifier evaluation require numeric predictors.
% Text/categorical variables are converted into numeric category codes.
% Numeric predictors are retained as numeric values.

Xnumeric = zeros(height(Xtable), width(Xtable));

for i = 1:width(Xtable)

    xi = Xtable{:, i};

    if iscellstr(xi) || isstring(xi) || ischar(xi)
```

```
    xi = categorical(xi);
  end

  if iscategorical(xi)
    Xnumeric(:, i) = double(xi);
  else
    Xnumeric(:, i) = double(xi);
  end
end

%% ============================================================
%  6. Remove rows with missing outcome
% ============================================================

validRows = ~ismissing(Ycat);
Xtable = Xtable(validRows, :);
Xnumeric = Xnumeric(validRows, :);
Ycat = Ycat(validRows);
Y = Y(validRows);

%% ============================================================
%  7. Remove invalid predictors
% ============================================================

% Remove predictors that are completely missing or have only one unique value.
% Such variables cannot contribute to classification.

varsToRemove = false(1, width(Xtable));
```

```
for i = 1:size(Xnumeric, 2)

   xi = Xnumeric(:, i);

   if all(isnan(xi))
      varsToRemove(i) = true;
   else
      xiNonMissing = xi(~isnan(xi));
      if numel(unique(xiNonMissing)) <= 1
         varsToRemove(i) = true;
      end
   end
end

removedVars = Xtable.Properties.VariableNames(varsToRemove);

Xtable(:, varsToRemove) = [];
Xnumeric(:, varsToRemove) = [];

fprintf('\nParticipants included: %d\n', height(Xtable));
fprintf('Candidate predictors after cleaning: %d\n', width(Xtable));

if ~isempty(removedVars)
   fprintf('\nRemoved variables because they were empty or had only one unique value:\n');
   disp(removedVars');
end

featureNames = Xtable.Properties.VariableNames;
```

```
%% ============================================================
%  8. Handle missing predictor values
% ============================================================

% Missing values are imputed using the median of each variable.
% In the final nested-CV pipeline, imputation should be fitted inside
% the training folds only.

for i = 1:size(Xnumeric, 2)

   xi = Xnumeric(:, i);

   if any(isnan(xi))
      medianValue = median(xi, 'omitnan');
      xi(isnan(xi)) = medianValue;
      Xnumeric(:, i) = xi;
   end
end

%% ============================================================
%  9. Standardize predictors
% ============================================================

% Standardization is important before LASSO because LASSO coefficients are
% affected by predictor scale.
%
% In the final nested-CV pipeline, mean and standard deviation should be
% estimated using training folds only.
```

```
mu = mean(Xnumeric, 1, 'omitnan');
sigma = std(Xnumeric, 0, 1, 'omitnan');

% Avoid division by zero.
sigma(sigma == 0) = 1;

Xz = (Xnumeric - mu) ./ sigma;

%% ============================================================
%  10. First stage: LASSO logistic regression
% ============================================================

% LASSO is used as an embedded feature-selection method.
% It shrinks less informative coefficients to zero.
% Here, binomial LASSO logistic regression is used because the outcome is binary.

rng(2026);

lassoCVFolds = 5;
lassoAlpha = 1;  % Alpha = 1 corresponds to LASSO. Alpha between 0 and 1 is Elastic Net.

fprintf('\nRunning LASSO logistic regression...\n');

[B, FitInfo] = lassoglm(Xz, Y, 'binomial', ...
   'Alpha', lassoAlpha, ...
   'CV', lassoCVFolds, ...
   'Standardize', false);

% Use the one-standard-error lambda for a more parsimonious model.
```

```
idxLambda = FitInfo.Index1SE;

lassoCoefficients = B(:, idxLambda);

lassoSelectedIdx = find(lassoCoefficients ~= 0);

% Fallback rule:
% If LASSO with Index1SE selects too few or zero features, use IndexMinDeviance.
if numel(lassoSelectedIdx) < 2

   fprintf('\nLASSO Index1SE selected fewer than 2 features. Using IndexMinDeviance instead.\n');

   idxLambda = FitInfo.IndexMinDeviance;
   lassoCoefficients = B(:, idxLambda);
   lassoSelectedIdx = find(lassoCoefficients ~= 0);
end

% Second fallback rule:
% If still no features are selected, select the top features by absolute coefficient
% from the minimum deviance solution.
if isempty(lassoSelectedIdx)

   fprintf('\nLASSO selected no features. Applying fallback based on absolute coefficients.\n');

   idxLambda = FitInfo.IndexMinDeviance;
   lassoCoefficients = B(:, idxLambda);

   [~, coefRank] = sort(abs(lassoCoefficients), 'descend');
   fallbackK = min(20, numel(coefRank));
```

```
    lassoSelectedIdx = coefRank(1:fallbackK);
end

lassoCandidateNames = featureNames(lassoSelectedIdx);
lassoCandidateCoefs = lassoCoefficients(lassoSelectedIdx);

fprintf('\nNumber of predictors retained after LASSO stage: %d\n', numel(lassoSelectedIdx));

%% ============================================================
%  11. Second stage: Genetic Algorithm settings
% ============================================================

% GA searches for the best feature subset among the LASSO-retained features.
% Each feature is represented as a binary variable:
% 1 = selected by GA
% 0 = not selected by GA.

nGAFeatures = numel(lassoSelectedIdx);

% GA reproducibility seed.
gaRandomSeed = 2026;
rng(gaRandomSeed);

% GA hyperparameters.
gaPopulationSize = 80;
gaMaxGenerations = 100;
gaCrossoverFraction = 0.80;
gaMutationRate = 0.05;
gaEliteCount = max(2, round(0.05 * gaPopulationSize));
```

```
gaMaxStallGenerations = 20;
gaFunctionTolerance = 1e-4;

% Fitness-function settings.
gaCVFolds = 5;
complexityPenalty = 0.01;

% Store settings for reporting.
gaSettingsTable = table( ...
  {'Genetic Algorithm'; ...
   'Binary chromosome'; ...
   'Population size'; ...
   'Maximum generations'; ...
   'Selection function'; ...
   'Crossover fraction'; ...
   'Mutation function'; ...
   'Mutation rate'; ...
   'Elite count'; ...
   'Maximum stall generations'; ...
   'Function tolerance'; ...
   'Random seed'; ...
   'Fitness criterion'; ...
   'Cross-validation folds in GA fitness'; ...
   'Complexity penalty'}, ...
  {'MATLAB ga'; ...
   '1 = selected, 0 = not selected'; ...
   num2str(gaPopulationSize); ...
   num2str(gaMaxGenerations); ...
   'Tournament selection'; ...
```

```
    num2str(gaCrossoverFraction); ...
    'Uniform mutation'; ...
    num2str(gaMutationRate); ...
    num2str(gaEliteCount); ...
    num2str(gaMaxStallGenerations); ...
    num2str(gaFunctionTolerance); ...
    num2str(gaRandomSeed); ...
    'Mean CV-AUC minus penalty for subset size'; ...
    num2str(gaCVFolds); ...
    ['AUC - ' num2str(complexityPenalty) ' * selected_features / LASSO_candidates']}, ...
   'VariableNames', {'GA_Setting', 'Value'});

fprintf('\nGA settings:\n');
disp(gaSettingsTable);

%% ============================================================
%  12. Prepare GA optimization problem
% ============================================================

% GA minimizes an objective function.
% Therefore, the objective is negative fitness:
% objective = - (mean CV-AUC - complexity penalty)

Xga = Xz(:, lassoSelectedIdx);
YgaCat = Ycat;

nvars = nGAFeatures;
intcon = 1:nvars;
lb = zeros(1, nvars);
```

```
ub = ones(1, nvars);

fitnessFunction = @(chromosome) hybridGAFitness( ...
    chromosome, ...
    Xga, ...
    YgaCat, ...
    positiveClass, ...
    gaCVFolds, ...
    complexityPenalty);

% GA options.
gaOptions = optimoptions('ga', ...
    'PopulationSize', gaPopulationSize, ...
    'MaxGenerations', gaMaxGenerations, ...
    'SelectionFcn', @selectiontournament, ...
    'CrossoverFraction', gaCrossoverFraction, ...
    'MutationFcn', {@mutationuniform, gaMutationRate}, ...
    'EliteCount', gaEliteCount, ...
    'MaxStallGenerations', gaMaxStallGenerations, ...
    'FunctionTolerance', gaFunctionTolerance, ...
    'Display', 'iter', ...
    'UseParallel', false);

%% ============================================================
%  13. Run Genetic Algorithm
% ============================================================

fprintf('\nRunning Genetic Algorithm feature selection...\n');
```

```
[bestChromosome, bestObjectiveValue, exitFlag, gaOutput] = ga( ...
   fitnessFunction, ...
   nvars, ...
   [], [], [], [], ...
   lb, ub, ...
   [], ...
   intcon, ...
   gaOptions);

bestChromosome = round(bestChromosome);
gaSelectedLocalIdx = find(bestChromosome == 1);

% Safety fallback:
% If GA selects no features, select the single strongest LASSO coefficient.
if isempty(gaSelectedLocalIdx)

   fprintf('\nGA selected no features. Applying fallback to strongest LASSO coefficient.\n');

   [~, strongestIdx] = max(abs(lassoCandidateCoefs));
   gaSelectedLocalIdx = strongestIdx;
end

gaSelectedOriginalIdx = lassoSelectedIdx(gaSelectedLocalIdx);

selectedFeatures = featureNames(gaSelectedOriginalIdx);
selectedCoefficients = lassoCoefficients(gaSelectedOriginalIdx);

%% ============================================================
%  14. Calculate final hybrid importance scores
```

```
% ============================================================

% Hybrid score is based on the absolute LASSO coefficient of GA-selected features.
% These scores are normalized for visualization.
%
% Interpretation:
% The GA determines whether a LASSO-retained feature is selected.
% The absolute LASSO coefficient provides a within-method importance ranking
% among the GA-selected features.

selectedScores = abs(selectedCoefficients);

% Remove zero-score features, if any.
positiveScoreIdx = selectedScores > 0;

selectedFeatures = selectedFeatures(positiveScoreIdx);
selectedCoefficients = selectedCoefficients(positiveScoreIdx);
selectedScores = selectedScores(positiveScoreIdx);

if isempty(selectedScores)
    error('All selected hybrid scores were zero. Please review LASSO and GA settings.');
end

selectedScoresNorm = selectedScores ./ max(selectedScores);

prettySelectedFeatures = makePrettyLabels(selectedFeatures);

resultsTable = table( ...
    selectedFeatures(:), ...
```

```
    prettySelectedFeatures(:), ...
    selectedCoefficients(:), ...
    selectedScores(:), ...
    selectedScoresNorm(:), ...
    'VariableNames', {'Original_Feature_Name', 'Readable_Feature_Label', 'LASSO_Coefficient',
'Hybrid_Importance_Score', 'Normalized_Hybrid_Importance_Score'});

fprintf('\nSelected Hybrid LASSO + GA features:\n');
disp(resultsTable);

%% ============================================================
%  15. Evaluate final selected subset using CV-AUC for reporting
% ============================================================

[finalMeanAUC, finalSDAUC] = evaluateSubsetAUC( ...
    Xz(:, gaSelectedOriginalIdx), ...
    Ycat, ...
    positiveClass, ...
    gaCVFolds);

gaSummaryTable = table( ...
    nGAFeatures, ...
    numel(selectedFeatures), ...
    -bestObjectiveValue, ...
    finalMeanAUC, ...
    finalSDAUC, ...
    exitFlag, ...
    gaOutput.generations, ...
    gaOutput.funccount, ...
```

```
    'VariableNames', {'LASSO_Candidate_Features', 'GA_Selected_Features', 'Best_GA_Fitness', 'Final_Selected_Subset_Mean_AUC', 'Final_Selected_Subset_SD_AUC', 'GA_ExitFlag', 'GA_Generations', 'GA_FunctionEvaluations'});

fprintf('\nGA optimization summary:\n');
disp(gaSummaryTable);

%% ============================================================
%  16. Save output tables
% ============================================================

outputFolder = fileparts(dataPath);

writetable(resultsTable, fullfile(outputFolder, 'Hybrid_LASSO_GA_Selected_Features.xlsx'));
writetable(gaSettingsTable, fullfile(outputFolder, 'Hybrid_LASSO_GA_Settings.xlsx'));
writetable(gaSummaryTable, fullfile(outputFolder, 'Hybrid_LASSO_GA_Summary.xlsx'));

lassoSettingsTable = table( ...
   {'LASSO model'; ...
    'Outcome distribution'; ...
    'Alpha'; ...
    'Cross-validation folds'; ...
    'Lambda selection rule'; ...
    'Selected Lambda'; ...
    'LASSO candidates retained'}, ...
   {'lassoglm'; ...
    'Binomial'; ...
    num2str(lassoAlpha); ...
    num2str(lassoCVFolds); ...
```

```
    'Index1SE preferred; IndexMinDeviance fallback if fewer than two features selected'; ...
    num2str(FitInfo.Lambda(idxLambda)); ...
    num2str(numel(lassoSelectedIdx))}, ...
   'VariableNames', {'LASSO_Setting', 'Value'});

writetable(lassoSettingsTable, fullfile(outputFolder, 'Hybrid_LASSO_Settings.xlsx'));

lassoCandidateTable = table( ...
   lassoCandidateNames(:), ...
   makePrettyLabels(lassoCandidateNames(:)), ...
   lassoCandidateCoefs(:), ...
   abs(lassoCandidateCoefs(:)), ...
   'VariableNames', {'Original_Feature_Name', 'Readable_Feature_Label', 'LASSO_Coefficient',
'Absolute_LASSO_Coefficient'});

writetable(lassoCandidateTable, fullfile(outputFolder, 'Hybrid_LASSO_Candidate_Features.xlsx'));

%% ============================================================
%  17. Produce readable bar chart
% ============================================================

[sortedScores, sortIdx] = sort(selectedScoresNorm, 'ascend');
sortedFeatures = selectedFeatures(sortIdx);
sortedLabels = makePrettyLabels(sortedFeatures);

figure('Color', 'w', 'Position', [100 100 1500 900]);

barh(sortedScores, 0.70);
```

```
yticks(1:numel(sortedLabels));
yticklabels(sortedLabels);

ax = gca;
ax.FontSize = 10;
ax.TickLabelInterpreter = 'none';

xlabel('Normalized Hybrid Importance Score', 'FontSize', 12);
title('Selected Features Ranked by Hybrid LASSO + GA', 'FontSize', 14, 'FontWeight', 'bold');

xlim([0, max(sortedScores) * 1.35]);

% Use five decimal places to avoid rounding small values to 0.00.
for i = 1:numel(sortedScores)
   text(sortedScores(i) + 0.015, i, sprintf('%.5f', sortedScores(i)), ...
      'VerticalAlignment', 'middle', ...
      'FontSize', 9);
end

grid on;
box on;

set(gcf, 'PaperPositionMode', 'auto');

saveas(gcf, fullfile(outputFolder, 'Hybrid_LASSO_GA_Selected_Features_BarChart.png'));
savefig(gcf, fullfile(outputFolder, 'Hybrid_LASSO_GA_Selected_Features_BarChart.fig'));

exportgraphics(gcf, fullfile(outputFolder,
'Hybrid_LASSO_GA_Selected_Features_BarChart_HighResolution.png'), 'Resolution', 300);
```

```
fprintf('\nDone.\n');
fprintf('Results saved in:\n%s\n', outputFolder);

%% ============================================================
%  18. Helper function: GA fitness
% ============================================================

function objectiveValue = hybridGAFitness(chromosome, X, Ycat, positiveClass, cvFolds, penaltyWeight)

  % GA minimizes this function.
  % The chromosome is a binary vector:
  % 1 = feature selected
  % 0 = feature not selected

  chromosome = round(chromosome);
  selectedIdx = find(chromosome == 1);

  % If no feature is selected, return a bad objective value.
  if isempty(selectedIdx)
    objectiveValue = 1e6;
    return
  end

  Xsubset = X(:, selectedIdx);

  try
    [meanAUC, ~] = evaluateSubsetAUC(Xsubset, Ycat, positiveClass, cvFolds);
```

```
    subsetPenalty = penaltyWeight * (numel(selectedIdx) / size(X, 2));

    fitness = meanAUC - subsetPenalty;

    % GA minimizes, so use negative fitness.
    objectiveValue = -fitness;

  catch
    % If model training fails, penalize this chromosome.
    objectiveValue = 1e6;
  end
end

%% ============================================================
%  19. Helper function: evaluate subset using CV-AUC
% ============================================================

function [meanAUC, sdAUC] = evaluateSubsetAUC(Xsubset, Ycat, positiveClass, cvFolds)

  cvp = cvpartition(Ycat, 'KFold', cvFolds);

  aucValues = nan(cvFolds, 1);

  for fold = 1:cvFolds

    trainIdx = training(cvp, fold);
    testIdx = test(cvp, fold);
```

```
    Xtrain = Xsubset(trainIdx, :);
    Ytrain = Ycat(trainIdx);

    Xtest = Xsubset(testIdx, :);
    Ytest = Ycat(testIdx);

    % Ensemble model is used inside GA fitness to evaluate subsets.
    % This keeps the fitness function consistent with the study focus on
    % ensemble-based predictive performance.
    mdl = fitcensemble(Xtrain, Ytrain, ...
        'Method', 'Bag', ...
        'NumLearningCycles', 100);

    [~, scores] = predict(mdl, Xtest);

    modelClassNames = cellstr(mdl.ClassNames);
    posCol = find(strcmp(modelClassNames, positiveClass));

    if isempty(posCol)
        error('Positive class not found in model class names.');
    end

    [~, ~, ~, aucValue] = perfcurve(Ytest, scores(:, posCol), categorical({positiveClass}));

    aucValues(fold) = aucValue;
end

meanAUC = mean(aucValues, 'omitnan');
sdAUC = std(aucValues, 'omitnan');
```

```
end

%% ============================================================
%  20. Helper function: readable labels
% ============================================================

function prettyLabels = makePrettyLabels(featureNames)

   prettyLabels = cell(size(featureNames));

   for j = 1:numel(featureNames)

      label = char(featureNames{j});

      label = strrep(label, '_', ' ');
      label = strrep(label, '-', ' ');

      label = regexprep(label, '([a-z])([A-Z])', '$1 $2');
      label = regexprep(label, '([A-Z])([A-Z][a-z])', '$1 $2');
      label = regexprep(label, '([0-9])([A-Za-z])', '$1 $2');
      label = regexprep(label, '([A-Za-z])([0-9])', '$1 $2');

      label = strrep(label, ' Of ', ' of ');
      label = strrep(label, ' In ', ' in ');
      label = strrep(label, ' The ', ' the ');
      label = strrep(label, ' And ', ' and ');
      label = strrep(label, ' For ', ' for ');

      label = strrep(label, 'P A', 'Physical Activity');
```

```
    label = strrep(label, 'PA', 'Physical Activity');
    label = strrep(label, 'B M I', 'BMI');

    label = strrep(label, 'C 8', 'C8');
    label = strrep(label, 'C8 Past Smoking', 'Past Smoking');
    label = strrep(label, 'C 8 Past Smoking', 'Past Smoking');

    label = strrep(label, 'Age of Smoking Init iation', 'Age of Smoking Initiation');
    label = strrep(label, 'Age of Smoking Cessation', 'Age of Smoking Cessation');
    label = strrep(label, 'Years Since Smoking', 'Years Since Smoking');
    label = strrep(label, 'Cigars Smoked Weekly', 'Cigars Smoked Weekly');

    label = strrep(label, 'Frequency of Eating Fruits', 'Frequency of Eating Fruits');
    label = strrep(label, 'Frequency of Eating Fruit', 'Frequency of Eating Fruits');
    label = strrep(label, 'Frequency of Eating Vegetables', 'Frequency of Eating Vegetables');
    label = strrep(label, 'Frequency of Eating Red Meat', 'Frequency of Eating Red Meat');
    label = strrep(label, 'Frequency of Eating Grilled Red Meat', 'Frequency of Eating Grilled Red
Meat');
    label = strrep(label, 'Frequency of Eating Chicken', 'Frequency of Eating Chicken');
    label = strrep(label, 'Frequency of Eating Grilled Chicken', 'Frequency of Eating Grilled
Chicken');

    label = strrep(label, 'place of Living', 'Place of Living');
    label = strrep(label, 'Place of Living', 'Place of Living');
    label = strrep(label, 'yearly Income', 'Yearly Income');
    label = strrep(label, 'occupation', 'Occupation');
    label = strrep(label, 'Residence', 'Residence');
    label = strrep(label, 'Marital Status', 'Marital Status');
    label = strrep(label, 'work yes No', 'Work Status');
```

```
    label = strrep(label, 'Hand Rolled Cigarettes Smoked Weekly', 'Hand-Rolled Cigarettes Smoked Weekly');

    label = strrep(label, 'Manufactured Cigarettes Smoked Weekly', 'Manufactured Cigarettes Smoked Weekly');

    label = strrep(label, 'Pipe Full of Tobacco Smoked Weekly', 'Pipe Full of Tobacco Smoked Weekly');


    label = strrep(label, 'No of Times Consumed Alcohol in the Past 12 Months', 'Alcohol Consumption Frequency in the Past 12 Months');

    label = strrep(label, 'Any Alcohol in the Past 12 Months', 'Any Alcohol Consumption in the Past 12 Months');

    label = strrep(label, 'Occasions of Drinking Alcohol', 'Occasions of Drinking Alcohol');

    label = strrep(label, 'Times of Drinking in the Past 30 Days', 'Times of Drinking in the Past 30 Days');

    label = strrep(label, 'Drinks in the Past 30 Days', 'Drinks in the Past 30 Days');

    label = strrep(label, 'Alcohol Consumption', 'Alcohol Consumption');

    label = strrep(label, 'Quit Alcohol for Health Problem', 'Quit Alcohol for Health Problem');


    label = strrep(label, 'Quit Smoking Last 12 Months', 'Quit Smoking in the Last 12 Months');


    label = strtrim(regexprep(label, '\s+', ' '));


    prettyLabels{j} = label;

  end

end
```