```
%CROSS_VALIDATION Simple k-fold cross-validation for DT / RF. function
meanAcc = cross_validation(X, y, modelType, params, kFolds)

  if nargin < 5
    kFolds = 5;
  end

  % Create partition for k-fold CV  cv
  = cvpartition(y, 'KFold', kFolds); acc
  = zeros(kFolds, 1);

  for k = 1:kFolds
    % Get indices for this fold
    trainIdx = training(cv, k);
    testIdx  = test(cv, k);

    % Split data according to indices
    Xtr = X(trainIdx, :);
    ytr = y(trainIdx);
    Xte = X(testIdx, :);
    yte = y(testIdx);

    switch lower(modelType)
      case 'dt'
        % Train a classification tree with specified max splits
        mdl = fitctree(Xtr, ytr, ...
          'MaxNumSplits', params.maxNumSplits);

        % Predict labels for validation fold
        yPred = predict(mdl, Xte);

      case 'rf'
        % Train a bagged ensemble for classification mdl
        = TreeBagger(params.numTrees, Xtr, ytr, ...
          'Method', 'classification', 'OOBPrediction', 'Off');

        % Same prediction but treeBagger.predict often returns cell arrays -> convert to numeric
        yPred = predict(mdl, Xte);
        if iscell(yPred)
          yPred = str2double(yPred);
        end

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

```
        % Compute fold accuracy
        acc(k) = mean(yPred == yte);
    end

    % Return mean accuracy across folds
    meanAcc = mean(acc);
end

%------------------------------------------------------------------------------
```

%METRICS Compute basic classification metrics for multi-class MNIST.

```
function metricsStruct = metrics(yTrue, yPred)

    % Ensure column vectors
    yTrue = yTrue(:);
    yPred = yPred(:);

    % Accuracy
    accuracy = mean(yTrue == yPred);

    % Confusion matrix
    classes = unique(yTrue);
    C = confusionmat(yTrue, yPred, 'Order', classes);

    % Per-class precision/recall/F1
    numClasses = numel(classes);
    precision  = zeros(numClasses, 1);
    recall     = zeros(numClasses, 1);   f1
    = zeros(numClasses, 1);

    % Compute metrics for each class for
    i = 1:numClasses
        tp = C(i, i);
         fp = sum(C(:, i)) - tp;
         fn = sum(C(i, :)) - tp;

        precision(i) = tp / max(tp + fp, eps);
        recall(i)    = tp / max(tp + fn, eps);
        f1(i)        = 2 * precision(i) * recall(i) / max(precision(i) + recall(i), eps);
    end

    % Package metrics into struct
    metricsStruct = struct();
    metricsStruct.accuracy  = accuracy;
```

```
    metricsStruct.confusion = C;
    metricsStruct.classes   = classes;
    metricsStruct.precision = precision;
    metricsStruct.recall    = recall;
    metricsStruct.f1        = f1;
end

%---------------------------------------------------------------------------
```

% LOAD_DATA Reads MNIST IDX files into MATLAB matrices.

```
function [XTrain, yTrain, XTest, yTest] = load_data()

    % File paths
    thisDir = fileparts(mfilename('fullpath'));
    baseDir = fullfile(thisDir, '..', 'data');

    trainImagesFile = fullfile(baseDir, 'train-images.idx3-ubyte');
    trainLabelsFile = fullfile(baseDir, 'train-labels.idx1-ubyte');
    testImagesFile  = fullfile(baseDir, 't10k-images.idx3-ubyte');
    testLabelsFile  = fullfile(baseDir, 't10k-labels.idx1-ubyte');

    % Verify files exist
    assert(isfile(trainImagesFile), 'File not found: %s', trainImagesFile);
    assert(isfile(trainLabelsFile), 'File not found: %s', trainLabelsFile);
    assert(isfile(testImagesFile),  'File not found: %s', testImagesFile);
    assert(isfile(testLabelsFile),  'File not found: %s', testLabelsFile);

    % Lode training data
    XTrain = load_images(trainImagesFile);
    yTrain = load_labels(trainLabelsFile);

    % Load test data
    XTest  = load_images(testImagesFile);
    yTest  = load_labels(testLabelsFile);

    fprintf('Loaded MNIST: %d train samples, %d test samples.\n', ...
        size(XTrain,1), size(XTest,1));
end

% Helper function to load images function
images = load_images(filename)
    fid = fopen(filename,'rb');
    if fid < 0, error('Could not open %s', filename); end

    magic = fread(fid,1,'int32',0,'ieee-be');
```

```
    if magic ~= 2051
        error('Invalid magic number in MNIST image file %s',filename);
    end

    numImages = fread(fid,1,'int32',0,'ieee-be');
    numRows   = fread(fid,1,'int32',0,'ieee-be');
    numCols   = fread(fid,1,'int32',0,'ieee-be');

    rawData = fread(fid, numImages*numRows*numCols, 'uint8');
    fclose(fid);

    rawData = reshape(rawData, numRows*numCols, numImages)';
    images = double(rawData);  % convert to double
end

% Helper function to load labels function
labels = load_labels(filename)
    fid = fopen(filename,'rb');
    if fid < 0, error('Could not open %s', filename); end

    magic = fread(fid,1,'int32',0,'ieee-be'); if
    magic ~= 2049
        error('Invalid magic number in MNIST label file %s',filename);
    end

    numLabels = fread(fid,1,'int32',0,'ieee-be');
    labels = fread(fid, numLabels, 'uint8');
    fclose(fid);
end

%-----------------------------------------------------------------------------

%PREPROCESS_DATA Basic preprocessing for MNIST pixels function
[XTrainProc, XTestProc] = preprocess_data(XTrain, XTest)

    fprintf('Preprocessing data (normalisation)...\n');

    XTrainProc = double(XTrain);
    XTestProc  = double(XTest);

    % scale pixels from [0,255] to [0,1]
    XTrainProc = XTrainProc / 255;
    XTestProc  = XTestProc  / 255;

    % visualize pixel intensity distribution
```

```
    figure;
    histogram(XTrainProc(:), 50);
    xlabel('Pixel intensity (rescaled to [0,1])');
    ylabel('Frequency');
    title('Distribution of rescaled pixel intensities');
    grid on;

    % normal standard distribution
    mu = mean(XTrainProc, 1);
    sigma = std(XTrainProc, 0, 1) + 1e-6;
    XTrainProc = (XTrainProc - mu) ./ sigma;
    XTestProc  = (XTestProc  - mu) ./ sigma;

    % visualize standardised pixel distribution
    figure;
    [f,xi] = ksdensity(XTrainProc(:));
    plot(xi,f,'LineWidth',2);
    xlabel('Pixel value (z-score)');
    ylabel('Density');
    title('Density of standardised pixel values');
    grid on;

end

%--------------------------------------------------------------------------

%TRAIN_DECISION_TREE Train a decision tree classifier with simple CV
function [dtModel, info] = train_decision_tree(XTrain, yTrain)

    fprintf('\n=== Training Decision Tree ===\n');

    % Hyperparameter grid
    hyperparamValues = [100, 200, 400, 600, 800];

    kFolds = 5;
    cvAccuracy = zeros(size(hyperparamValues));

    % Time the cross validation
    tCV = tic;
    for i = 1:numel(hyperparamValues)
        maxSplits = hyperparamValues(i);
        fprintf('  CV for MaxNumSplits = %d ...\n', maxSplits);

        params.maxNumSplits = maxSplits;
        cvAccuracy(i) = cross_validation(XTrain, yTrain, 'dt', params, kFolds);
```

```
  end
  cvTimeSeconds = toc(tCV);

  % Choose best hyperparameter
  [~, bestIdx] = max(cvAccuracy);
  bestMaxSplits = hyperparamValues(bestIdx);
  fprintf('Best MaxNumSplits = %d (CV accuracy = %.3f)\n', ...
    bestMaxSplits, cvAccuracy(bestIdx));

  % Time the final training
  tTrain = tic;
  dtModel = fitctree(XTrain, yTrain, 'MaxNumSplits', bestMaxSplits);
  trainTimeSeconds = toc(tTrain);

  % Fill info structure
  info.modelName        = 'Decision Tree';
  info.hyperparamName   = 'MaxNumSplits';
  info.hyperparamValues  = hyperparamValues;
  info.cvAccuracy       = cvAccuracy;
  info.bestHyperparam   = bestMaxSplits;
  info.cvTimeSeconds     = cvTimeSeconds;
  info.trainTimeSeconds  = trainTimeSeconds;
end

%-----------------------------------------------------------------------------

%TRAIN_RANDOM_FOREST Train a Random Forest (bagged trees) with simple CV
function [rfModel, info] = train_random_forest(XTrain, yTrain)

  fprintf('\n=== Training Random Forest ===\n');

  % Hyperparameter grid
  numTreesGrid = [50, 100, 200];
  kFolds = 5;
  cvAccuracy = zeros(size(numTreesGrid));

  % Time the cross validation
  tCV = tic;
  for i = 1:numel(numTreesGrid)
    params.numTrees = numTreesGrid(i);
    fprintf('  CV for NumTrees = %d ...\n', params.numTrees); cvAccuracy(i) =
    cross_validation(XTrain, yTrain, 'rf', params, kFolds);
  end
  cvTimeSeconds = toc(tCV);
```

```
  % Choose best hyperparameter
  [~, bestIdx] = max(cvAccuracy);
  bestNumTrees = numTreesGrid(bestIdx);
  fprintf('Best NumTrees = %d (CV accuracy = %.3f)\n', ...
    bestNumTrees, cvAccuracy(bestIdx));

  % Time the final training
  tTrain = tic;
  rfModel = TreeBagger(bestNumTrees, XTrain, yTrain, ...
             'Method', 'classification', ...
             'OOBPrediction', 'On');
  trainTimeSeconds = toc(tTrain);

  % Fill info structure
  info.modelName        = 'Random Forest';
  info.hyperparamName    = 'NumTrees';
  info.hyperparamValues  = numTreesGrid;
  info.cvAccuracy       = cvAccuracy;
  info.bestHyperparam    = bestNumTrees;
  info.cvTimeSeconds     = cvTimeSeconds;
  info.trainTimeSeconds  = trainTimeSeconds;
end

%-----------------------------------------------------------------------------

%EVALUATE_MODEL Evaluate model on train and test sets, save confusion matrix.
function results = evaluate_model(model, XTrain, yTrain, XTest, yTest, modelName, confusionFigPath)

  fprintf('\n=== Evaluating %s ===\n', modelName);

  % Predictions on train data set
    yPredTrain = local_predict(model, XTrain);
    trainMetrics = metrics(yTrain, yPredTrain);
  fprintf('%s - Train accuracy: %.4f\n', modelName, trainMetrics.accuracy);

  % Predictions on test data set
  yPredTest = local_predict(model, XTest);
  testMetrics = metrics(yTest, yPredTest);
  fprintf('%s - Test accuracy:  %.4f\n', modelName, testMetrics.accuracy);

  % Plot confusion matrix for test set
  figure('Visible','off');
  confusionchart(testMetrics.confusion, string(testMetrics.classes));
  title(sprintf('Confusion Matrix - %s (Test)', modelName));
```

```
    if ~isempty(confusionFigPath)
        [figDir, ~, ~] = fileparts(confusionFigPath); if
        ~exist(figDir, 'dir'); mkdir(figDir); end
        saveas(gcf, confusionFigPath);
    end
    close(gcf);

    % Package results
    results = struct();
    results.modelName     = modelName;
    results.trainAcc       = trainMetrics.accuracy; =
    results.testAcc        testMetrics.accuracy;
    results.metricsTest  = testMetrics;
    results.metricsTrain = trainMetrics;
end

% ------------------------------

% Local helper for predictions
function yPred = local_predict(model, X) if
    isa(model, 'TreeBagger')
        yPred = predict(model, X);
        if iscell(yPred)
            yPred = str2double(yPred);
        end
    else
        % Assume it's a ClassificationTree or similar
        yPred = predict(model, X);
    end
end

%------------------------------------------------------------------------------

%PLOT_RESULTS Plot accuracy comparison and hyperparameter tuning curves.
function plot_results(dtResults, rfResults, dtInfo, rfInfo, accFigPath, hyperFigPath)

    % Accuracy comparison
    figure('Visible','off');
    modelNames = {dtResults.modelName, rfResults.modelName};
    testAcc    = [dtResults.testAcc,  rfResults.testAcc];

    bar(testAcc);
    set(gca, 'XTickLabel', modelNames);
    ylabel('Test accuracy');
```

```
    title('Decision Tree vs Random Forest on MNIST');

    for i = 1:numel(testAcc)
        text(i, testAcc(i), sprintf('%.3f', testAcc(i)), ...
            'HorizontalAlignment','center', 'VerticalAlignment','bottom');
    end

    if ~isempty(accFigPath)
        [figDir, ~, ~] = fileparts(accFigPath);
        if ~exist(figDir, 'dir'); mkdir(figDir); end
        saveas(gcf, accFigPath);
    end
    close(gcf);

    % Hyperparameter curves
    figure('Visible','off');
    hold on;

    plot(dtInfo.hyperparamValues, dtInfo.cvAccuracy, '-o', 'DisplayName', dtInfo.modelName);
    plot(rfInfo.hyperparamValues, rfInfo.cvAccuracy, '-s', 'DisplayName', rfInfo.modelName);

    xlabel('Hyperparameter value');
    ylabel('CV accuracy');
    title('Hyperparameter Tuning');
    legend('Location','best');
    grid on;

    if ~isempty(hyperFigPath)
        [figDir, ~, ~] = fileparts(hyperFigPath); if
        ~exist(figDir, 'dir'); mkdir(figDir); end
        saveas(gcf, hyperFigPath);
    end
    close(gcf);
end

%------------------------------------------------------------------------------

% IN3300 Project – Decision Tree vs Random Forest on MNIST
clear; clc; close all;
rng(42);

% filepath management   thisFile
= mfilename('fullpath');
[thisDir,~,~] = fileparts(thisFile);
```

```
addpath(fullfile(thisDir, 'utils'));

% load_data.m
[XTrain, yTrain, XTest, yTest] = load_data();

% preprocess_data.m
[XTrainProc, XTestProc] = preprocess_data(XTrain, XTest);

%train_decision_tree.m %
train_random_forest.m
  [dtModel, dtInfo] = train_decision_tree(XTrainProc, yTrain);
  [rfModel, rfInfo] = train_random_forest(XTrainProc, yTrain);

% time for training and CV
fprintf('\n=== Training times ===\n');
fprintf('Decision Tree  - train: %.3f s, CV: %.3f s\n', ...
   dtInfo.trainTimeSeconds, dtInfo.cvTimeSeconds);
fprintf('Random Forest  - train: %.3f s, CV: %.3f s\n', ...
   rfInfo.trainTimeSeconds, rfInfo.cvTimeSeconds);

% print hyperparameter tuning results
fprintf('\n=== Hyperparameter Tuning Results ===\n');

fprintf('\nDecision Tree (MaxNumSplits):\n');
for i = 1:numel(dtInfo.hyperparamValues)
   fprintf('  MaxNumSplits = %d: CV Accuracy = %.4f\n', ...
      dtInfo.hyperparamValues(i), dtInfo.cvAccuracy(i));
end

fprintf('\nRandom Forest (NumTrees):\n');
for i = 1:numel(rfInfo.hyperparamValues)
   fprintf('  NumTrees = %d: CV Accuracy = %.4f\n', ...
      rfInfo.hyperparamValues(i), rfInfo.cvAccuracy(i));
end

% evaluate_model.m
dtResults = evaluate_model(dtModel, XTrainProc, yTrain, ...
                    XTestProc,  yTest,  'Decision Tree', ...
                    fullfile('results','confusion_dt.png'));

rfResults = evaluate_model(rfModel, XTrainProc, yTrain, ...
                    XTestProc,  yTest,  'Random Forest', ...
                    fullfile('results','confusion_rf.png'));

% plot_results.m
```

```
resultsDir = fullfile('results');
if ~exist(resultsDir, 'dir'); mkdir(resultsDir); end

plot_results(dtResults, rfResults, dtInfo, rfInfo, ...
   fullfile('results','accuracy_comparison.png'), ...
   fullfile('results','hyperparameter_plot.png'));
```