% 1. Create a simple dataset consisting of two variables that are linearly
% related. Randomly divide the dataset into halves; use one half to fit a
% linear model and use the other half to test that model. Compute the R^2
% for the fitted model on the training data. Then compute the R^2 for the
% fitted model on the testing data. Finally, compute the R^2 of the true
% model (i.e. the model that generated the data) on the testing data.
% Repeat this training/testing process many times, each time with a different
% random split of the data. Finally, summarize and visualize the results.

% define some constants
numdata = 20*2;  % even number of data points just to make splitting easy
numsimulations = 1000;

% generate some data
% this is the input variable.
x = randn(1,numdata);
% this function describes the true model.
modelfun = @(x) 0.7*x + 2;
% this is the output variable.
y = feval(modelfun,x) + randn(1,numdata);

% run simulations
R2train = zeros(1,numsimulations);  % R^2 of fitted model on training data
R2test = zeros(1,numsimulations);   % R^2 of fitted model on testing data
R2true = zeros(1,numsimulations);   % R^2 of true model on testing data
for rep=1:nsimulations
    % randomly split data into halves
    order = randperm(length(x));
    trainindex = order(1:end/2);
    testindex = order(end/2+1:end);

    % construct regressor matrix and fit model
    X = [x(trainindex)' ones(length(trainindex),1)];
    w = inv(X'*X)*X'*y(trainindex)';

    % evaluate R^2 of fitted model on training data
    R2train(rep) = computeR2(X*w,y(trainindex)');

    % construct regressor matrix for testing data
    X = [x(testindex)' ones(length(testindex),1)];

    % evaluate R^2 of fitted model on testing data
    R2test(rep) = computeR2(X*w,y(testindex)');

    % evaluate R^2 of true model on testing data
    R2true(rep) = computeR2(feval(modelfun,x(testindex)'),y(testindex)');

% visualize results
figure; hold on;
mn = [mean(R2train) mean(R2test) mean(R2true)];
se = [std(R2train)/sqrt(length(R2train)) ...
    std(R2test)/sqrt(length(R2test)) ...
    std(R2true)/sqrt(length(R2true))];
errorbar(1:3,mn,se,'r-');
ax = axis; axis([0.5 3.5 ax(3:4)]);
set(gca,'XTick',1:3,'XTickLabel',{'Training' 'Testing' 'True'});
xlabel('');
ylabel('R^2');
title('Training R^2 overestimates testing R^2');

% Notice that the accuracy of the fitted model on the training data
% is higher than the accuracy of the fitted model on the testing data.
% This reflects the fact that the accuracy of a model on training data
% overestimates the true accuracy of the model.
%
% Also, notice that the accuracy of the fitted model on the testing data
% is lower than the accuracy of the true model on the testing data. This
% reflects the fact that although the fitted model is in some sense the
% best we can do given the training data, the fitted model is still not
% optimal.

% 2. Create a simple dataset consisting of two variables. Use k-fold cross-
% validation to compare the accuracy of a linear model against the accuracy of
% a quadratic model. Does one model outperform the other? Compute a p-value
% for the null hypothesis that the predictions of the two models are no
% different from each other.
% let's fix the random number seeds so that we get the same results each time
rand('state',0);
randn('state',0);

% generate some data
x = 2 + randn(1,200);
y = 1.5*x.^2 + x + 1 + 2*randn(1,200);

% fit linear model using 20-fold cross-validation
[R2_linear,predictions_linear] = ...
crossvalidatelinearmodel([x' ones(length(x),1)],y',20);

% fit quadratic model using 20-fold cross-validation
[R2_quadratic,predictions_quadratic] = ...
crossvalidatelinearmodel([x'.^2 x' ones(length(x),1)],y',20);

% compute error bars on the R^2 values
R2_linear_errorbars = ...
prctile(bootstrp(1000,@computeR2,predictions_linear,y'),[16 84]);
R2_quadratic_errorbars = ...
prctile(bootstrp(1000,@computeR2,predictions_quadratic,y'),[16 84]);
% the following is necessary for the conventions of errorbar.m
R2_linear_errorbars = (R2_linear - R2_linear_errorbars) .* [1 -1];
R2_quadratic_errorbars = (R2_quadratic - R2_quadratic_errorbars) .* [1 -1];

% compute p-value for the null hypothesis that the predictions of the two
% models are interchangeable
predaggregate = [predictions_linear' predictions_quadratic'];
dataaggregate = [y y];
dist = zeros(1,1000);
for p=1:1000
    ii1 = ceil(rand(1,length(predictions_linear))*length(predaggregate));
    ii2 = ceil(rand(1,length(predictions_quadratic))*length(predaggregate));
    dist(p) = computeR2(predaggregate(ii1),dataaggregate(ii1)) - ...
        computeR2(predaggregate(ii2),dataaggregate(ii2));
end
pval = sum(abs(dist)>abs(R2_quadratic-R2_linear)) / length(dist);

% visualize results
figure; set(gcf,'Position',[100 100 800 400]);
% first panel showing data and the cross-validated predictions
subplot(1,2,1); hold on;
    h1 = scatter(x,y,'r.');
    h2 = scatter(x,predictions_linear,'go');
    h3 = scatter(x,predictions_quadratic,'bo');
    legend([h1 h2 h3],['Data' 'Linear model' 'Quadratic model'],'Location','Best');
xlabel('x');
ylabel('y');
title('Cross-validated predictions');
% second panel showing the R^2 values
subplot(1,2,2); hold on;
    bar(1:2,[R2_linear R2_quadratic]);
function f = computeR2(model, data)
    f = 100*(1-sum((data-model).^2)/sum((data-mean(data)).^2));
end

function [R2, predictions] = crossvalidatelinearmodel(X, y, numfolds)
    % fit a linear model relating X to y using k-fold cross-validation.
    % the number of folds is given by numfolds, and we randomly assign data % points to folds.
end
% we return:
% <R2> as the R^2 between the model predictions and the data
% <predictions> as a column vector with the model predictions
%
calculate some constants
numdata = length(y);

% figure out cross-validation scheme
% first, evenly distribute data points into folds
xval = mod(1:numdata,numfolds) + 1;
% then, randomly assign data points to the folds
xval = xval(randperm(numdata));

% do the model fitting
% this will hold the model prediction for each data point
predictions = zeros(size(y));
for p=1:numfolds

% figure out the indices of the data points to use for
% testing and the data points to use for training
testindex = find(xval==p);
trainindex = setdiff(1:numdata,testindex);

% estimate the free parameters of the model
% using the training data
X0 = X(trainindex,:);
w = inv(X0'*X0)*X0'*y(trainindex);

% evaluate the prediction of the fitted model
% for the testing data, and record the results
X0 = X(testindex,:);
predictions(testindex) = X0*w;
end

% finally, compute the R^2 between all of the model predictions and the data
R2 = computeR2(predictions,y);